VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra Informatiky

Generováni Word dokumentů na základě šablon

Generating of Word Documents Based on Templates

2011 **Dávid Siro**

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2011 .....

Chcel by som poďakovať všetkým, ktorí mi pri písaní tejto diplomovej práce pomáhali, hlavne Ing. Romanovi Hrdému, vedúcemu diplomovej práce, za odbornú pomoc a Ing. Janovi Martinovičovi, Ph.D., môjmu konzultantovi na Katedre informatiky VŠB-TUO za cenné rady a pripomienky.

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 2. Května 2011 .....

Abstrakt

Táto práca sa zaoberá problematikou generovania Word dokumentov na základe šablón. Realizácia prebiehala v spolupráci s firmou E LINKX a.s., kde bolo zanalyzované firemné riešenie používané na generovanie dokumentov. Na základe tejto analýzy bolo navrhnuté a implementované riešenie nové. Nový systém je založený na štandarde Open XML. Šablóny sú tvorené Word dokumentmi a pomocou softvérovej komponenty sú pretransformované na výsledný dokument. Práca zahrňuje návrh značkovacieho jazyka pre šablóny, návrh samotnej komponenty a taktiež popisuje jej nasadenie v reálnom prostredí.

Kľúčové slová: Microsoft Office, Word, šablóna, generovanie dokumentov, Open XML

Abstract

This thesis deals with issue of template-based Word documents generating. Realization of this thesis ran in cooperation with E LINKX a.s., where a company's solution used for documents generating was analyzed. Based on this analysis a new solution was designed and implemented. A new solution is based on Open XML standard. Templates are represented by Word documents, which are transformed into resulting documents using this software component. Thesis includes design of template's markup language, design of software component itself and it also describes deployment of the solution into production environment.

Keywords: Microsoft Office, Word, template, generating of documents, Open XML

Zoznam použitých symbolov a skratiek

ANTLR	ANother Tool for Language Recognition
COM	Component Object Model
DLR	Dynamic Language Runtime
MS	Microsoft
OOXML	Office Open XML
OPC	Open Packaging Convetion
XML	Extended Markup Language

Obsah

1 Úvod

Problematika reportov a zostáv je dôležitou oblasťou tvorby firemných informačných systémov. Úlohou reportov je poskytovať v užívateľsky prehľadnej forme relevantné dáta a informácie. Reporty majú mnoho rolí, či už prinášajú nové znalosti a slúžia k podpore rozhodovania, alebo tvoria požadované výstupné zostavy ako sú napr. faktúry alebo rôzne iné dokumenty, sú neoddeliteľnou súčasťou takmer každého informačného systému.

Jedným z najrozšírenejších kancelárskych nástrojov je v súčasnosti balík MS Office, ktorého formáty široko akceptované. S príchodom verzie 2007 prináša MS Office nový formát dokumentov založený jazyku XML, ktorý prináša celú radu výhod. Mať preto možnosť automatizovaného generovania reportov (resp. dokumentov) vo formátoch MS Office môže znamenať výraznú úsporu času a finančných prostriedkov.

Táto diplomová práca vznikla v spolupráci s firmou E LINKX a.s. [5] a jej hlavným cieľom bolo navrhnúť a implementovať komponentu umožňujúcu generovanie dokumentov na základe šablón vo formáte MS Office Word, ktorá by bola použiteľná v informačných systémoch dodávaných touto firmou.

V rámci firmy už existuje riešenie tejto problematiky, ktoré bolo analyzované v druhej kapitole.

Na základe analýzy z druhej kapitoly boli špecifikované požiadavky na novú komponentu, ktorej analýza a návrh boli spracované v tretej, resp. štvrtej kapitole. Obsah týchto kapitol zahrňuje návrh štruktúra komponenty, značkovacieho jazyka pre šablóny, popis štandardu Office Open XML a ďalších relevantných vlastností.

Piata kapitola sa zaoberá detailmi implementácie a v šiestej kapitole je popísaná integrácia hotovej komponenty do jedného z produktov firmy E LINKX.

2 Analýza súčasného stavu generovania Word dokumentov

Úlohou analyzovanej komponenty je generovanie dokumentov na základe užívateľský definovaných šablón. Tento pomerne jednoduchý koncept ukrýva niekoľko problémov. Táto kapitola sa zaoberá analýzou riešenia generovania dokumentov, používaného vo firme E LINKX.

2.1 Implementácia

Generovanie Word dokumentov je implementované ako samostatná, znovu použiteľná komponenta (samostatný .dll súbor), postavená na technológii .NET a implementovaná v jazyku C#. Dokument sa generuje na základe definovanej šablóny a dátového zdroja (ADO.NET). Finálny dokument vzniká nahradzovaním značiek v šablóne automatizovaním aplikácie Microsoft Word 2003 ako COM komponenty [1], ktorá poskytuje rozhranie pre manipuláciu s dokumentom. Generovanie teda neprebieha úpravami priamo súboru dokumentu, ale sprostredkovane za pomoci MS Word API.

2.2 Šablóny

Šablóna je reprezentovaná Word dokumentom, v ktorom sú použité špeciálne značky, ktoré sú spracovávané a nahradzujú sa za dáta z dátového zdroja. Značka začína a končí znakom # (napr. *#ZNACKA#*). Značky so štýlovateľné, tzn. po nahradení majú dáta rovnaký štýl ako mala značka.

Z pohľadu funkčnosti môžeme značky rozdeliť na tri typy tzv. *jednoduché dáta, riadkové dáta* a *repeater*.

Ako *dátový zdroj* je využívaná technológia ADO.NET, konkrétne objekty typu DataTable (Výpis 1).

```
public string Parse(string file, DataTable replacement, List<DataTable>
rowsTables, List<DataTable> rowsInLineTables)
{
   //Parsovanie sablony
}
```

VÝPIS 1: ROZHRANIE SÚČASNEJ KOMPONENTY .

Vo výpise 1 je vidieť že vstupmi pre generovanie dokumentov je meno súboru s požadovanou šablónou, ďalšími parametrami sú:

- *replacement* – jedná sa o tzv. jednoduché dáta, objekt DataTable obsahuje len jeden riadok,
- *rowsTables* – riadkové dáta,
- *rowsInLineTables* – tzv. repeater, špeciálny prípad riadkových dát.

Výstupom komponenty je cesta k dočasnému súboru s vygenerovaným dokumentom.

Jednotlivé typy vstupných parametrov sú podrobnejšie rozobrané v ďalších kapitolách.

2.3 Značky

Jednoduché dáta sú v texte šablóny reprezentované pomocou konvencie #NAZOV_STLPCA#. Táto značka sa nahradzuje hodnotou prvého riadku a príslušného stĺpca(podľa názvu uvedeného medzi znakmi #).

Riadkové dáta reprezentujú množinu riadkov (objektov), ktoré sa majú vykresliť (každý na samostatnom riadku). V šablóne je toto realizované pomocou tabuľky s jedným riadkom, v ktorej sú jednotlivé stĺpce reprezentované pomocou konvencie #NAZOV_TABULKY.NAZOV_STLPCA# (šablóna teda môže obsahovať viacero nezávislých častí z riadkovými dátami).

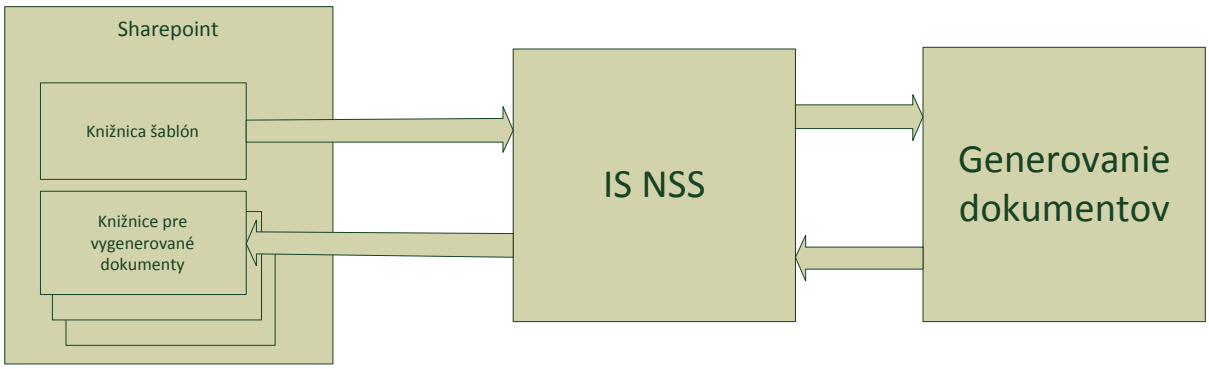
Pri spracovávaní šablóny sa prejdú všetky riadky objektu DataTable s príslušným názvom, a pre každý jej riadok vzniká nový riadok v v danej tabuľke v dokumente, pričom sa nahradia značky hodnotami z príslušných stĺpcov. Výsledkom je teda n-riadková tabuľka.

Repeater je obdoba predchádzajúcej značky, rozdiel je v tom že, kým v minulom prípade vznikal pre každý riadok zo zdrojového objektu DataTable jeden riadok v danej tabuľke , v tomto prípade sa generuje celá zdrojová množina za sebou v rámci jedného odseku. Značka teda tvorí samostatný odsek v dokumentu a jednotlivé polia sú reprezentované pomocou konvencie #REPEATER_NAZOV_TABULKY.NAZOV_STLPCA# (opäť je tu možnosť viacerých nezávislých “repeaterov” v rámci jednej šablóny).

2.4 Nasadenie komponenty

Komponentu využíva Nejvyšší správní soud (NSS) ČR v rámci svojho informačného systému. Pomocou tejto komponenty sú generované dokumenty NSS. Komponenta je používaná v prostredí MS Sharepoint (šablóny a generované dokumenty sú ukladané do dokumentových knižníc Sharepointu).

Z dôvodu zakúpenej licencie, ktorá bola zakúpená v čase implementácie je vyžadovaná výsledná verzia dokumentov vo formáte Word 2003.



OBRÁZOK 1: GENEROVANIE DOKUMENTOV V RÁMCI IS NSS

2.5 Nevýhody

Nevýhody tohto riešenie možno zhrnúť do niekoľkých bodov:

- rýchlosť – spracovanie dokumentu pomocou COM komponenty Word je pomalé a a pamäťovo náročné, COM objekt nie vždy korektne uvoľňuje pamäťové prostriedky,
- nutnosť mať nainštalovaný MS Word na serveri kde prebieha generovanie – pri takomto využití Wordu nestačí len jedna licencia,
- bezpečnosť – pri práci s COM objektom musí dôjsť k preautentizácii užívateľa na silné práva,
- škálovateľnosť – aplikácia MS Word je primárne navrhnutá pre prácu jedného užívateľa a nie pre serverové spracovanie mnohých požiadavkov,
- nemožnosť podmieneného vykresľovania – prezentačná logika nie je súčasťou šablóny, nutnosť robiť úpravu dát v kóde(pri zmene formátovania nutná kompilácia...), nie je oddelená aplikačná logika od prezentačnej,
- vykresľovanie hierarchických informácii – v súčasnej komponente nie je možné vykresľovať hierarchické štruktúry,
- dátový zdroj – ako dátový zdroj len ADO.NET DataTable, v prípade že aplikácia využíva vlastnú dátovú vrstvu(napr. objekty), nie je ich možné použiť priamo ako dátový zdroj pre dokument,
- *repeater* a *Riadkové dáta* - v podstate rovnaká funkcionalita, možno nahradiť jedným prvkom,
- nie je možné použiť rovnakú značku nahradzujúcu stĺpec z tabuľky jednoduchých dát viackrát v dokumente, v prípade potreby viacnásobného použitia sa musí pridať odpovedajúci počet stĺpcov do zdrojovej tabuľky,
- v rámci *riadkových dát* nie je možné štýlovať časti opakovaných dát.

3 Analýza nového riešenia

Táto kapitola zahrňuje aj funkčné a nefunkčné požiadavky na nové riešenie, vyplývajúce z analýzy starého riešenia. V ďalších podkapitolách je popísaná objektovo-orientovaná analýza novej komponenty, analýza funkčnosti značiek ako aj možné alternatívy k riešeniu v podobe hotových komponent tretích strán.

3.1 Požiadavky

Funkčné požiadavky na novú softvérovú komponentu sa oproti pôvodnej verzii na základe analýzy existujúcej komponenty a postrehov z praktického využívania mierne zmenili, no koncept samostatnej komponenty umožňujúcej generovanie Word dokumentov zostáva nezmenený.

Funkčné požiadavky	
Názov	**Popis**
Generovanie Word dokumentov na základe šablón	Bude existovať sada značiek umožňujúcich strojové spracovanie dokumentu. Základná značka umožňuje vloženie poľa(property) dátového zdroja.
Šablóna musí byť užívateľský modifikovateľná	Šablóna je tvorená Word dokumentom s vopred špecifikovanými značkami, užívateľ si môže túto šablónu sám modifikovať.
Dátový zdroj šablóny je objekt	Dáta v šablóne budú viazané na vlastnosti objektu(dátového zdroja), ako dátový zdroj bude možné použiť aj objekty typu ADO.NET DataTable.
Prezentačná logika bude súčasťou šablóny	Pomocou špeciálnych značiek bude možne testovať rôzne podmienky a na základe toho upraviť vykresľovanie dát.
Vykresľovanie hierarchický informácii	Komponenta bude umožňovať vykresľovanie hierarchických štruktúr tvorených kolekciami objektov
Vloženie dokumentu	Bude existovať značka umožňujúca vložiť externý dokument.
Štýlovanie dát	Dátam v šablóne bude možné nastaviť štýl a formátovanie
Generovanie hlavičky a päty	Značky bude možné umiestniť aj do hlavičky a paty
Vloženie obrázku	Na určené miesto v dokumente bude možné vložiť a formátovať obrázok
Podpora elektronického podpisovania dokumentov	Už vygenerované dokumenty bude možné digitálne podpisovať(podpora X.509

	certifikátov)
Vloženie obrázku, text do existujúceho dokumentu	Do vygenerovaného dokumentu bude možné dodatočne vložiť textové pole alebo obrázok

TABUĽKA 1: FUNKČNÉ POŽIADAVKY

Nefunkčné požiadavky vychádzajú z obmedzení vyplývajúcich z analýzy v kapitole 2 a používanej vývojovej platformy vo firme E LINKX a.s.

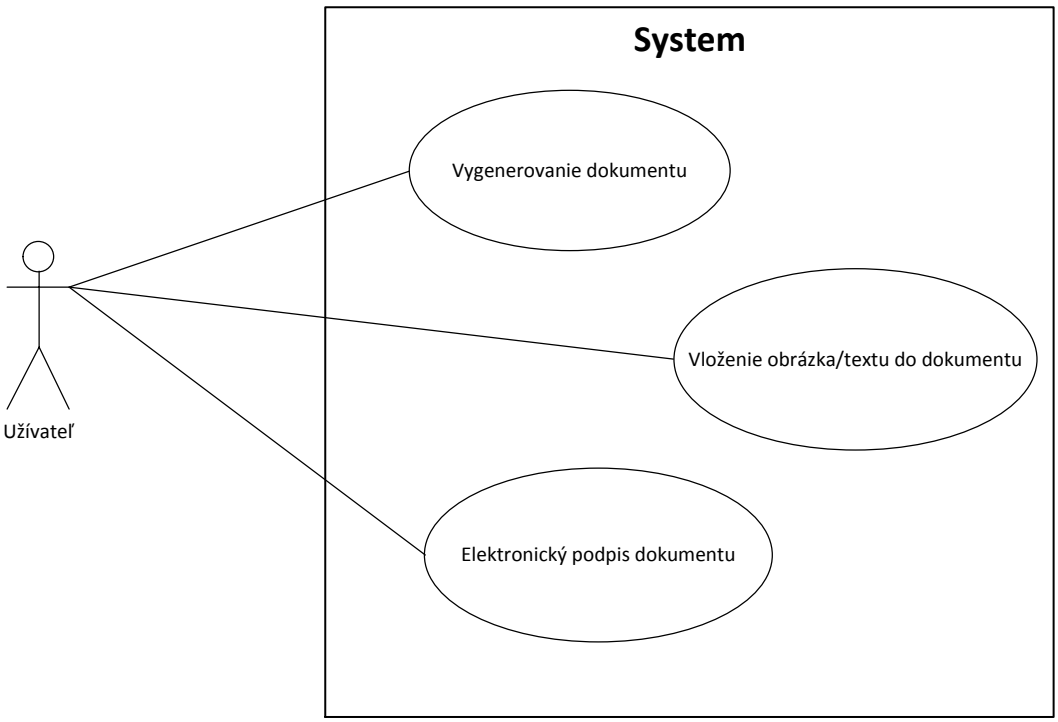
Nefunkčné požiadavky	
Názov	**Popis**
Implementácia pomocou technológie .NET a jazyka C#	
Spätná kompatibilita zo starými šablónami	
Generovanie dokumentov v rôznych verziách Wordu, možnosť vygenerovať ja PDF dokument	Najnižšia podporovaná verzia musí byť minimálne Word 2003.
Zrýchlenie generovania	
Komunikácia z prostredím MS Sharepoint	Ukladanie/načítavanie dokumentov z dokumentových knižníc.
Možnosť generovania prostredníctvom webovej služby	Generovanie dokumentov prístupné cez rozhranie webovej služby, umožňujúce jednoduchú škálovateľnosť (cluster).

TABUĽKA 2: NEFUKNČNÉ POŽIADAVKY

3.1.1 Use Case diagram

Na základe use case diagramu (Obrázok 2) je vidieť že s systémom bude pracovať len jeden aktér. - Týmto aktérom (na use case diagrame je zovšeobecňujúco nazvaný *Užívateľ*) môže byť buď externý systém požadujúci vygenerovanie dokumentu, iná komponenta, alebo programátor využívajúci túto komponentu.

Samotná komponenta má len tri prípady použitia. *Vygenerovanie dokumentu* je tvorí základnú funkcionalitu, teda generovanie dokumentov, *Elektronický podpis dokumentu* a *Vloženie obrázka/textu do dokumentu* sú samostatné prípady použitia, ktoré pracujú nad vygenerovanými dokumentmi(príp. akýmikoľvek inými dokumentmi) (viď Tabuľka 1).



OBRÁZOK 2: USE CASE DIAGRAM NOVEJ KOMPONENTY

3.2 Hotové produkty/iné prístupy k riešeniu

Na trhu existuje viacero riešení ponúkajúcich možnosť tvorby reportov vo formáte Microsoft Word. Jedným z nich je Microsoft SQL Server Reporting Services [13].

Tento produkt predstavuje akúsi nadstavbu pre MS SQL Server, pričom poskytuje serverovú platformu pre doručovanie reportov v rôznych formátoch. Paradoxne, ako dátový zdroj môže byť použitý nielen MS SQL Server ale aj rôzne iné databázy(Oracle, DB2, Hyperion...). Reporty je možné navrhovať pomocou špeciálnej aplikácie nazvanej Report Builder.

SQL Server Reporting Services je robustné riešenie, ponúkajúce široké možnosti tvorby, manažovanie a doručovania reportov.

Protikladom robustnosti SQL Server Reporting Services môžu byť komponenty firmy *Softartisans* [6], ktoré sa zameriavajú na serverové spracovanie formátov balíka MS Office. Produkt OfficeWriter je navrhnutý na skutočné serverové nasadenie. Pomocou tejto komponenty je možné generovať dokumenty vo formátoch Word alebo Excel. Dizajn reportov prebieha pomocou aplikácii MS Office(Word, resp. Excel). OfficeWriter podporuje tak binárne formáty, ako aj formáty založené na OOXML.

3.3 Analýza vlastného riešenia

V rámci analýzy riešenia budú uvedené triedny a sekvenčné diagramy, ako aj popis značiek potrebných k splneniu funkčných požiadavkov.

3.3.1 Značky

Samotná šablóna bude tvorená značkami ktoré predstavujú bloky dokumentu, ktoré budú spracovávané generátorom dokumentov. Ku každej značke je bude možné definovať atribúty ktoré budú určovať dodatočné alebo špecifické vlastnosti pre danú značku. Spôsob implementácie značiek v rámci dokumentu je uvedený v kapitole Návrh.

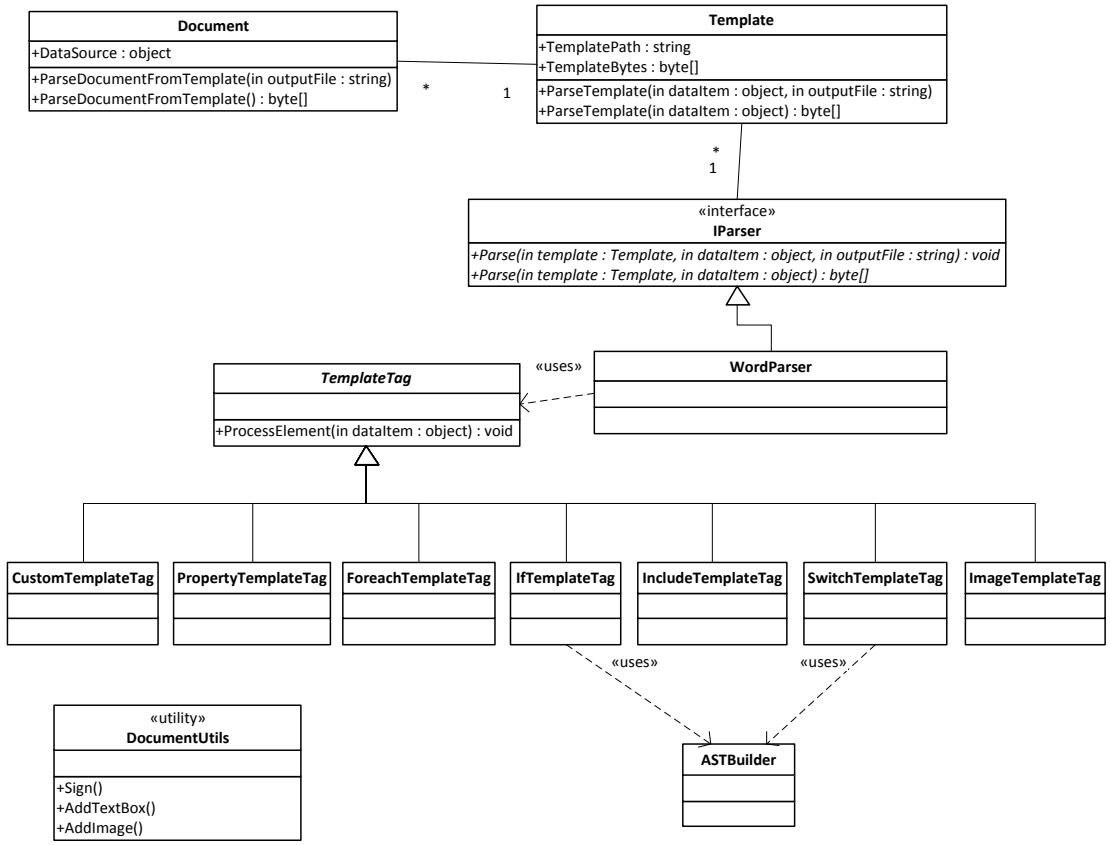
Na základe funkčných požiadavkov bude šablóna tvorená nasledujúcimi značkami:

Značka	Popis	Atribúty
property	vráti hodnotu vlastnosti (property) objektu v dátovom zdroji. Meno vlastnosti je definované obsahom značky	formatString – nepovinný umožní nastaviť formátovací reťazec
foreach	prechádza zadanú kolekciu, pre každý prvok opakuje blok ktorý je obsahom značky	in – názov kolekcie, ktorá sa bude prechádzať
if	podmienené vykresľovanie časti šablón. Môže obsahovať buď priamo blok ktorý sa má zobraziť splnení podmienky alebo samostatné značky *then* a *else*.	expression – booleovský výraz ktorý sa má vyhodnotiť. Bude definovaná gramatika, určujúca prípustné výrazy.
then	súčasť značky *if*, ak je prítomný, musí byť prítomná aj značka *else*. Reprezentuje časť dokumentu ktorá sa má zobraziť pri splnení podmienky	
else	reprezentuje časť dokumentu ktorá sa má zobraziť pri nesplnení podmienky	
switch	podmienené formátovanie na základe viacerých možných hodnôt výrazu	exp – výraz ktorý sa ma vyhodnotiť, gramatika bude vychádzať z gramatiky použitej pri značke if, výraz nemusí byť booleovský
case	súčasť značky *switch*, označuje konkrétnu hodnotu vyhodnocovaného výrazu	value – hodnota výrazu
default	súčasť značky *switch*, označuje predvolenú hodnotu	
include	vloží iný dokument	src – meno vlastnosti dátového zdroja ktorý obsahuje cestu k súboru file – cesta k súboru(umožňuje vložiť statický dokument –

		cesta je definovaná priamo v šablóne)
image	vloží obrázok z externého zdroja, obsah elementu definuje vlastnosť dátového zdroja, ktorá obsahuje cestu k obrázku	path – cesta k súboru, v prípade že obsah elementu je prázdny
customtag	umožňuje implementovať vlastnú funkcionalitu	

TABUĽKA 3: ZNAČKY

3.3.2 Triedny diagram



OBRÁZOK 3: TRIEDNY DIAGRAM - ANALÝZA

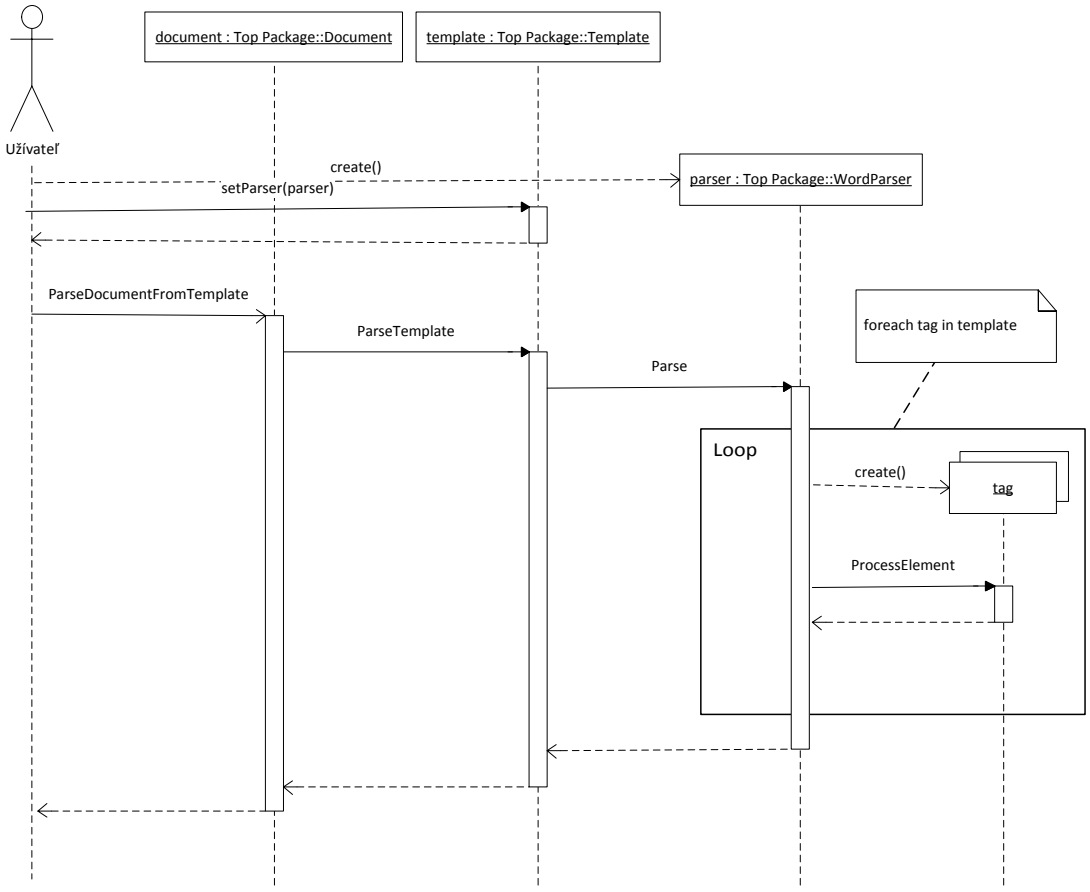
Model systému je navrhnutý všeobecne, umožňuje implementáciu generovania rôznych druhov dokumentov, nie je teda viazaný len na dokumenty MS Word (Obrázok 3).

Generovaný dokument je reprezentovaný triedou *Document* ktorá zapuzdruje celé generovanie. Trieda vystavuje dve metódy, jednu pre uloženie vygenerovaného dokumentu na disk, a druhú, ktorá vráti výsledok ako pole bajtov.

Jadrom systému je trieda *Template* ktorá reprezentuje šablónu dokumentu. Interne je šablóna tvorená buď poľom bajtov (*TemplateBytes*) alebo cestou k súboru so šablónou (*TemplatePath*). Samotné spracovanie šablóny už má na starosti trieda implementujúca rozhranie *IParser*. Je teda možné implementovať viacero rozhraní pre rôzne formáty šablón (HTML, RTF atď.). V rámci tejto diplomovej práce bolo implementované spracovanie šablóny pre dokumenty MS Word.

Trieda *WordParser* implementuje spracovanie šablóny pre dokumenty MS Word, pričom využíva triedy dediace z abstraktnej triedy *TemplateTag*, ktoré implementujú špecifickú funkcionalitu pre jednotlivé značky.

Pre jednoduchosť budem v ďalšom texte pod pojmom *parser* uvádzať inštanciu triedy *WordParser*.



OBRÁZOK 4: SEKVENČNÝ DIAGRAM GENEROVANIA DOKUMENTU

Špecifickú funkcionalitu má trieda *ASTBuilder*, ktorá zapuzdruje funkčnosť pre lexikálnu a syntaktickú analýzu jazyka použitého na tvorbu výrazov pre značky *if* a *switch*. Trieda vytvára a

vyhodnocuje syntaktický strom pre daný výraz. Gramatika a popis vyhodnocovania výrazov sú popísane v kapitole Návrh.

Prípady použitia *Vloženie obrázka/textu do dokumentu* a *Elektronický podpis dokumentu* sú implementované pomocou statických metód v pomocnej triede *DocumentUtils*.

3.3.3 Sekvenčný diagram generovania dokumentu

Princíp spracovania šablóny je znázornený sekvenčný diagramom (Obrázok 4). Užívateľ vytvorí inštanciu rozhrania *IParser*(v našom prípade je to inštancia triedy *WordParser*) a asociuje ju s objektom šablóny. Objektu triedy *Dokument* je potrebné nastaviť dátový zdroj(vlastnosť *DataSource*) a šablónu z ktorej sa má generovať.

Samotný parser prechádza šablónu a pre každú nájdenú značku inicializuje zodpovedajúcu triedu, ktorá implementuje jej funkcionalitu.

4 Návrh

V tejto kapitole bude na základe výsledkov analýzy navrhnutá výsledná komponenta. Pre implementáciu bolo zvolená platforma .NET a jazyk C# (používaná platforma vo firme E LINKX).

4.1 Office Open XML

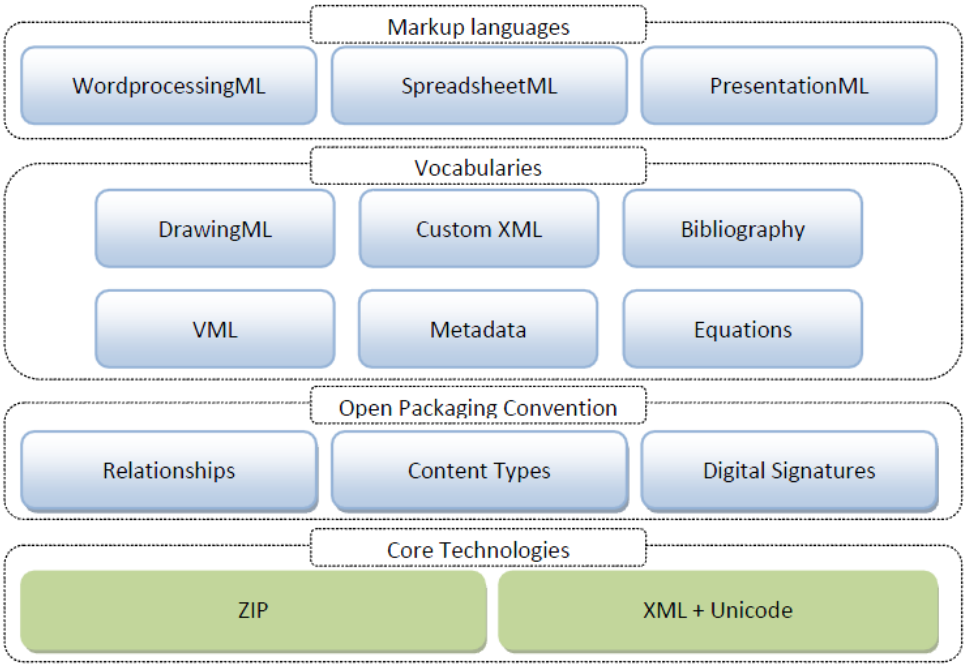
Ako formát šablón a predvolený formát vygenerovaných dokumentov bol zvolený formát Word dokumentov používaný od verzie Word 2007 (*docx*), založený na štandarde Office Open XML (OOXML) [9]. OOXML bolo štandardizované organizáciou ECMA po označením ECMA-376 a bolo prvý krát použité v balíku Microsoft Office 2007. Tento štandard, ktorý zahrňuje špecifikáciu pre viacero formátov dokumentov rodiny Microsoft Office, ponúka oproti pôvodnými binárnym formátom viacero výhod:

- Otvorenosť – formát je založený na štandardoch ZIP a XML, takže je univerzálne prístupný naprieč rôznymi platformami. Špecifikácia formátov a schémy sú dostupné pod licenciou bez poplatkov.
- Interoperabilita – výmena dát medzi MS Office a externými aplikáciami je zjednodušená. Aplikácie môžu pristupovať k dátam vnútri dokumentov, resp. vytvárať nové dokumenty bez nutnosti používať aplikácie MS Office pomocou štandardných nástrojov na manipuláciu s XML.
- Robustnosť – OOXML formáty sú navrhnuté robustnejšie oproti predchádzajúcim verziám, tým pádom znižujú riziko straty informácii v dôsledku poškodeného súboru.
- Efektívnosť – formát využíva ZIP kompresiu k ukladaniu dokumentov. Veľkosť výsledných dokumentov je do 75% menšia než u porovnateľných binárnych dokumentov.
- Bezpečnosť - otvorenosť OOXML formátov umožňuje vyššiu bezpečnosť. V rámci dokumentov je jednoduché identifikovať a odstrániť citlivé informácie a taktiež je možné identifikovať potencionálne nebezpečný obsah.

Kontajnerom pre OOXML dokument je ZIP archív, v ktorom sa nachádzajú rôzne časti dokumentu(XML súbory, obrázky, webové odkazy) prepojené vzájomnými väzbami. Špecifikácia formátu je rozdelená na viacero časti (Obrázok 5).

Jedna z vrstiev OOXML je definovaná pomocou tzv. *Open Packaging Convetion* (OPC). OPC je štandard (ECMA 376-20) ktorý definuje ako sú OOXML dokumenty ukladané v ZIP balíčku [7], pričom sa jedná o obecnú špecifikáciu, ktorá umožňuje vytváranie vlastných formátov. Tento štandard definuje vzťahy medzi jednotlivými logickými časťami dokumentu.

V rámci balíčku je každý súbor (XML, obrázok, video, atď.) časťou dokumentu (*document part*). Medzi jednotlivými časťami, ako aj medzi balíčkom a časťou sú definované väzby (*relationship*). Tieto väzby umožňujú vytvoriť požadovanú štruktúru dokumentu, zabezpečiť integritu dokumentu, zistiť obsah jednotlivých odkazovaných častí v rámci dokumentu, pristupovať k obsahu a radu iných výhod. Väzby sú previazané s adresárovou štruktúrou, ak teda zmeníme štruktúru archívu, musíme pre korektné zobrazenie dokumentu aktualizovať aj definície väzieb.



OBRÁZOK 5: VRSTVY OOXML [3]

Napríklad v prípade nového Word dokumentu má balíček predvolené väzby na tri časti - hlavnú časť dokumentu (*Main document part*), čo je XML súbor s obsahom dokumentu, tzv *Core properties part* a *Extended properties part*. Tieto časti môžu mať väzby na ďalšie časti, napr. hlavný dokument môže obsahovať väzby na obrázky a pod.

Ďalšou vrstvou OOXML štandardu sú značkovacie jazyky (*ML – markup languages*). Tieto jazyky popisujú XML značky pomocou ktorých sa vytvárajú jednotlivé dokumenty balíka Microsoft Office. Sadu značkovacích jazykov štandardu OOXML tvorí:

- WordprocessingML – popisuje značkovací jazyk pre textové procesory (Obrázok 6),
- SpreadsheetML - popisuje značkovací jazyk pre tabuľkové procesory,
- PresenationML - popisuje značkovací jazyk pre prezentácie.

Na základe vlastností OOXML bude generovanie dokumentov postavené na tejto technológii, teda aj formát šablón a vygenerovaných dokumentov bude *docx* (Word 2007 a vyššie). Výhodou tohto riešenia je prístupnosť dokumentov pomocou štandardných nástrojov- možnosť spracovávať šablónu ako XML súbor, odpadá nutnosť mať nainštalovaný MS Word ako aplikáciu k spracovaniu dokumentov, čo výrazne uľahčuje serverové spracovanie dokumentov. Odpadá nutnosť spracovávať binárny formát.

```
<w:p w:rsidR="00D70FF1" w:rsidRPr="002328FC" w:
    <w:pPr>
        <w:rPr>
            <w:lang w:val="cs-CZ" />
        </w:rPr>
    </w:pPr>
    <w:proofErr w:type="spellStart" />
    <w:r>
        <w:rPr>
            <w:lang w:val="cs-CZ" />
        </w:rPr>
        <w:t>Hello</w:t>
    </w:r>
    <w:proofErr w:type="spellEnd" />
    <w:r>
        <w:rPr>
            <w:lang w:val="cs-CZ" />
        </w:rPr>
        <w:t xml:space="preserve"> Word!</w:t>
    </w:r>
</w:p>
```

OBRÁZOK 6: UKÁŽKA WORDPROCESSINGML

Medzi hlavné výhody patrí takisto zrýchlenie generovania. Spracovanie XML šablóny je totiž rýchlejšie ako automatizácia Wordu pomocou COM komponenty, ktorá si vyžaduje náročnú réžiu.

Vďaka OOXML je zároveň jednoduché zabezpečiť integritu a validitu dokumentov – šablóny je možné validovať ako štandardne XML súbory voči zodpovedajúcim schémam.

Dôležitou požiadavkou bola však aj možnosť vygenerovať dokument starších verzii MS Word, prípadne iné formáty. Keďže po spracovaní šablóny bude výsledný formát dokumentu tiež *docx* je nutné navrhnúť mechanizmu konvertovania dokumentov do požadovaných formátov.

4.2 Značkovací jazyk pre šablóny

Pod pojmom značkovací jazyk, sa rozumie jazyk, pomocou ktorého bude možné do šablóny zapisovať značky a ach atribúty navrhnuté v analýze riešenia. Na základe požadovaných vlastností bude značkovanie založené na technológii CustomXML, ktorá je súčasťou štandardu OOXML.

V rámci tejto kapitoly sú rozobrané princípy na ktorých stojí značkovací jazyk šablón, detailný popis funkčnosti jednotlivých značiek je súčasťou prílohy C.

CustomXML je súčasť štandardu OOXML, ktorý umožňuje pridať užívateľské XML značky do dokumentu v rámci WordProcessingML. Užívateľ môže definovať vlastnú XML schému, a v nej definované značky požívať v dokumentoch.

OBRÁZOK 7: CUSTOMXML ELEMENT V DOKUMENTE

Vo výslednom XML súbore (Obrázok 7) je customXML značka definovaná elementom *<w:customXml* a nie priamo xml značkou z definovanej schémy [8].

CustomXML značky z hľadiska elementov, ktoré obaľujú delíme na tzv. *inline-level, block-level, cell-level* a *row-level.*

Inline-level customXML značky sa môžu nachádzať kdekoľvek v rámci textu dokumentu, a označujú rozsah textu v dokumentu v rámci nejakého nadradeného prvku (najčastejšie odsek, alebo bunka tabuľky). Oproti tomu *block-level* špecifikuje prítomnosť customXML elementu obaľujúceho jednu alebo viac blokových štruktúr (odseky, tabuľky atď.) *Cell-level* a *row-level* elementy označujú prítomnosť customXML okolo bunky, resp. riadka tabuľky.

CustomXML elementy teda predstavujú ideálnu možnosť ako definovať sémantické informácie v rámci dokumentov. Skutočnosť že ich možno definovať akoukoľvek validnou XML schémou zabezpečuje jednoduchou formou integritu výslednej šablóny. Možnosť značkovať časti tabuliek pomocou *cell-level* *row-level* elementov zasa dovoľuje do generovania dokumentov zahrnúť aj tieto časti. Vďaka zabudovanej podpore pre customXML značky sa dajú šablóny efektívne vytvárať v rámci aplikácii MS Word, nie je nutné implementovať vlastný dizajnér šablón. Značky používané v šablónach sú definované XML schémou (Príloha A). Detailný rozbor CustomXML značiek je súčasťou [8].

4.2.1 CustomXML a patenty

Aj keď je technológia CustomXML ideálny základ pre značkovací jazyk šablón, jej zachovania v produktov MS Office v budúcnosti môže byť otázne. V priebehu implementácie riešenia vyšlo najavo že Microsoft ako autor tejto technológie porušuje patent spoločnosti i4i. Na základe rozhodnutia súdu platnom na území USA, musela byť odstránená táto funkcionalita s aplikácie Word 2007 [12]. Verzie Wordu používané vo zvyšku sveta nie sú týmto rozhodnutím dotknuté, Microsoft však prestal podporovať CustomXML v aplikácii Word 2010 globálne.

V súčasnosti je možné šablóny naďalej navrhovať pomocou aplikácie Word 2007. Vzhľadom na časové možnosti bola dokončená implementácia komponenty pomocou technológie CustomXML.

Súčasťou ďalšieho vývoja komponenty môže byť implementácia alternatívneho značkovacieho jazyka, prípadne vývoj samostatného dizajnéra šablón, čím by sa proces tvorby šablón stal nezávislým na aplikácii MS Word.

4.3 Dátové zdroje

Medzi definované funkčné požiadavky patrí schopnosť komponenty pracovať s dátovým zdrojom vo forme jednoduchého objektu, ako aj s ADO.NET objektmi (DataTable). Jedným z prístupov ako získavať hodnoty členských premenných je prístup k nim pomocou reflexie . Tento prístup je výhodný u klasických objektov, kde členské premenné obsahujú hodnoty ktoré chceme vypisovať do výsledného dokumentu. V prípade ADO.NET DataTable je však situácia komplikovanejšia.

Objekty typu DataTable predstavujú tabuľku so stĺpcami a riadkami ktorá obsahuje určité dáta. V prípade že chceme pristúpiť k hodnote v určitom stĺpci a riadku, *DataTable* vystavuje kolekciu *Rows* kde môžeme pristupovať k jednotlivým riadkom a následne k hodnotám stĺpcov v danom riadku (Výpis 2).

```
DataTable table = new DataTable("Table");

//naplnenie hodnotami

object value = table.Rows[0]["col1"];
```

VÝPIS 2:PRÍSTUP K HODNOTÁM OBJEKTU DATATABLE

V prvom prípade sú vlastnosti ktoré chcem vypisovať v dokumente priamo verejnými členmi objektu, v druhom prípade sú však definované kolekciou riadkov a stĺpcov. Je síce možné pristupovať k týmto hodnotám reflexívne, ale tento prístup má jednu nevýhodu.

Predstavme si že máme šablónu do ktorej chceme vypísať obsah vlastnosti dátového zdroja s názvom *Name.* V prípade klasického objektu zistíme pomocu reflexie hodnotu jeho vlastnosti *Name* a vypíšeme ju do výsledného dokumentu. Ak však použijeme ako dátový zdroj objekt typu *DataTable*, museli by sme k reflexívne pristupovať najprv ku kolekcii *Rows,* a až potom by sme mohli pristúpiť k hodnote stĺpca *Name*. Znamenalo by to že pri zmene dátového zdroja šablóny, by sme museli zmeniť aj spôsob odkazovania sa na vlastnosti dátového zdroja, aj keď požadované dáta by ostali rovnaké, len by sa zmenila ich forma.

.NET Framework ponúka na tento problém riešenia v rámci namespace *System.ComponentModel*.

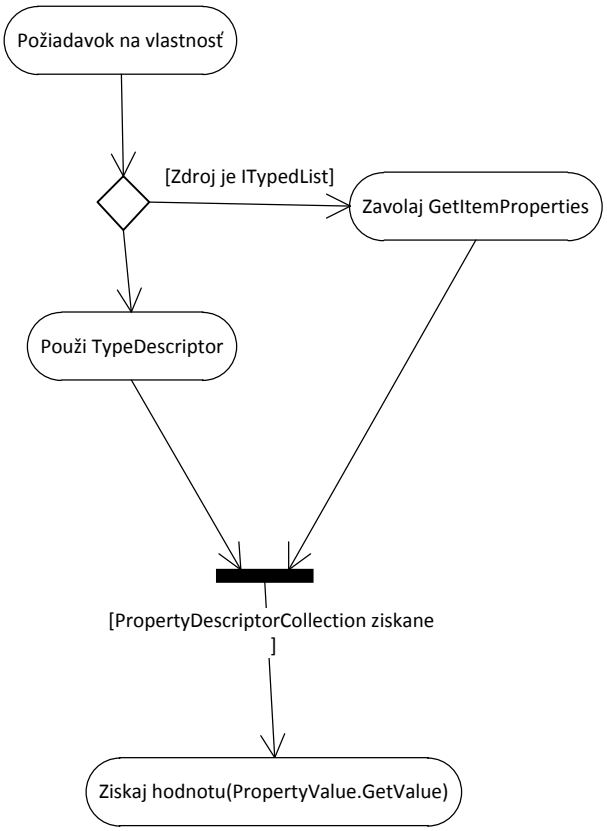
4.3.1 System.ComponentModel namespace

Tento namespace poskytuje sadu tried, ktoré sú používané k implementácii správania sa komponent v rámci .NET aplikácii. Zahŕňa rôzne triedy a rozhrania, nás však budú zaujímať tie, ktoré slúžia na abstrakciu viazania (binding) vlastností dátových zdrojov.

Komponentou sa v tomto prípade myslí aj dátový zdroj našej šablóny. K získaniu informácii o komponente ako sú atribúty, vlastnosti, udalosti slúži trieda *TypeDescriptor*. Táto trieda poskytuje rozšíriteľný inšpekčný mechanizmus pre komponenty.

Vyšetrovanie vlastností komponenty prebieha pomocou statickej metódy *GetProperties* ktorá pre danú inštanciu komponenty(v našom prípade to bude dátový zdroj), vráti kolekciu objektov triedy *PropertyDescriptor*(tzv. *PropertyDescriptorCollection*). Pre každú vlastnosť komponenty existuje jedna inštancia tejto triedy. *PropertyDescriptor* obsahuje užitočnú metódu *GetValue* pomocou ktorej môžeme získať hodnotu danej vlastnosti.

V prípade že chceme pracovať vlastnosťami ktoré sú uchovávane interným mechanizmom komponenty, môžeme použiť rozhranie *ITypedList*(toto rozhranie implementuje napr. aj trieda *DataView*). Toto rozhranie definuje metódu *GetItemProperties* vracajúcu opäť známu kolekciu *PropertyDescriptorCollection*. V prípade že chcem teda pristupovať k vlastnostiam komponenty ktorá implementuje *ITypedList*, nepoužívame na získanie kolekcie *PropertyDescriptor*-ov triedu *TypeDescriptor* ale zavoláme priamo metódu tohto rozhrania. Tento proces je znázornený aktivitným diagramom (Obrázok 8).

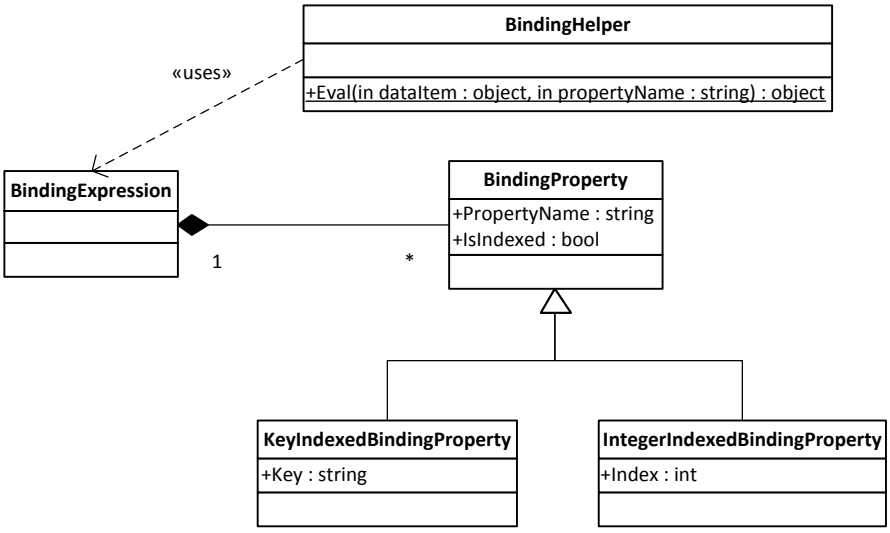


OBRÁZOK 8: INŠPEKCIA VLASTNOSTÍ DÁTOVÉHO ZDROJA

Rozhranie *ITypedList* a trieda *PropertyDescriptor* poskytujú unifikovaný a štandardný spôsob ako jednotným spôsobom pristupovať k vlastnostiam dátových zdrojov rôznych typov.

Ďalším, z pohľadu aplikácie dôležitým rozhraním z tohto namespace, je rozhranie *IListSource*. Toto rozhranie je používané vtedy, ak sa dátový zdroj skladá kolekcie prvkov ktorú chceme vyhodnotiť, ale neimplementuje rozhranie *IList* na prístup k nim. Toto rozhranie vystavuje metódu *GetList(),* ktorá vráti kolekciu spomínaných prvkov. Vlastnosť *ContainsListCollection* určuje či prvky vrátenej kolekcie sú typu *IList.*

Typickým príkladom je trieda *DataSet*, ktorá obsahuje kolekciu tabuliek (*DataTable*). *DataSet* implementuje *IListSource* tak že vráti kolekciu objektov *DataView*, čo je vlastne predvolený pohľad na príslušnú tabuľku.



OBRÁZOK 9:TRIEDNY DIAGRAM PRE PRÍSTUP K VLASTNOSTIAM DÁTOVÉHO ZDROJA

4.3.2 Trieda BindingHelper

Trieda *BindingHelper* poskytuje pomocné statické metódy pre prístup k vlastnostiam dátových zdrojov. Funkcionalita tejto triedy je využívaná značkami, ktoré potrebujú prístup k hodnotám vlastností, okrem značiek ktoré využívajú k svojej funkcionalite gramatiku na spracovanie výrazov (*If*, *Switch*). Statická štruktúra tejto časti je znázornená triednym diagramom (Obrázok 9).

Pre dátové zdroje podporované komponentou platia nasledujúce pravidlá:

- Dátovým zdrojom môže byť jednoduchý objekt, objekt ADO.NET typu *DataTable* alebo *DataSet*, alebo všeobecne ľubovoľný objekt implementujúci štandardné rozhranie *System.ComponentModel.ITypedList.*

- K vlastnostiam objekt sa pristupuje cez ich názvy – v prípade ADO.NET typov sú to názvy príslušných stĺpcov.
- Názvy vlastností môžu byť hierarchické, oddeľované pomocou bodky, napr. *Customer.Address.City*, a zároveň môžu obsahovať index založený na celom čísle alebo kľúči(reťazci), napr. *Collection["key"]* alebo *MyProperty[0]*. Viac násobné indexovanie (napr. *Property[0][0]*) nie je podporované.
- V prípade že je dátovým zdrojom *DataSet* pristupuje sa k vlastnostiam pomocou konvencie nezadáva index riadka, vtedy sa implicitne berie prvý riadok tabuľky (napr. *MyTable.Column1*).
- V prípade že je dátovým zdrojom len objekt typu *DataTable*, je možné pristupovať priamo k stĺpcom len pomocou ich názvov, hodnoty sa automatický berú z prvého riadka tabuľky.

Na triednom diagrame(Obrázok 9) je vidieť, že k ohodnocovaniu vlastností dátového zdroja je nutné zostaviť inštanciu triedy *BindingExpression*. Po zavolaní metódy *Eval* sa korektný vstupný reťazec(podľa hore uvedených konvencií) prevedie na kolekciu objektov typu *BindingProperty* a tieto sú vyhodnocované oproti dátovému zdroju (Výpis 3).

```
public static object Eval(object DataItem, string propertyName)
{
    try
    {
        BindingExpression expression = BindingExpression.Create(propertyName);
        object value = DataItem;
        for (int i = 0; i < expression.Properties.Count; i++)
        {
            value = GetValue(expression.Properties[i], value);
        }
        return value;

    }
    catch (Exception)
    {
        throw new ParserException("Cannot evaluate property: " + propertyName);
    }
}
```

VÝPIS 3:VYHODNOCOVANIE VÝRAZOV VOČI DÁTOVÉMU ZDROJU

Vyhodnotenie prebieha tak, že algoritmus prechádza kolekciu *Properties*, pričom prvá inštancia *BindingProperty* je vyhodnotená oproti pôvodnému dátovému zdroju šablóny a ďalšie inštancie sú vyhodnotené oproti hodnote z predchádzajúceho kroku.

K vyhodnocovaniu každej inštancie *BindingProperty* je interne používaná metóda *GetProperty* (Výpis 4).

Metóda je rozdelená na tri časti, každá testuje či dátový zdroj implementuje príslušné rozhrania a podľa toho získava správnu hodnotu danú parametrom *dataMember* (v komentároch sú uvádzané pre jednoduchosť a lepšiu názornosť názvy ADO.NET tried ktoré tieto rozhrania implementujú).

Najskôr sa testuje implementácia rozhrania *IListSource*, kde môžu nastať dva prípady. Prvý prípad (objekt implementuje *ITypedList* a zároveň platí že kolekcia vrátených objektov je typu *IList*) vyjadruje napr. situáciu keď z inštancie *DataSet*-u chceme získať tabuľku z konkrétnym názvom. V druhom prípade sa vráti prvý objekt z danej kolekcie a získa sa hodnota požadovanej vlastnosť (typicky objekt *DataTable* z ktorého získavame hodnotu príslušného stĺpca v prvom riadku).

```
private static object GetProperty(object dataSource, string dataMember)
{
    IListSource source = dataSource as IListSource;
    if (source != null)
    {
        if(source.ContainsListCollection && source.GetList() is ITypedList)
        {
             //vetva pre DataSet
        }
        else
        {
            IEnumerator enumerator = source.GetList().GetEnumerator();
            if(enumerator.MoveNext())
            {
                 //vetva DataTable – vráti prvý riadok

                object item = enumerator.Current;
                PropertyDescriptor descriptor =
                   TypeDescriptor.GetProperties(item).Find(dataMember, true);
                return descriptor.GetValue(item);
            }
        }
    }

    if(dataSource is ITypedList && dataSource is IEnumerable)
    {
       //vetva DataView – vráti prvý riadok

    }

    //Vetva pre jednoduchy objekt

    PropertyDescriptor propertyDescriptor =
         TypeDescriptor.GetProperties(dataSource).Find(dataMember, true);

    return propertyDescriptor.GetValue(dataSource);
}
```

VÝPIS 4: ZÍSKAVANIE HODNÔT BINDINGPROPERTY

V treťom prípade dátový zdroj implementuje *ITypedList* a zároveň *IEnumerable.* Funkcionalita je rovnaká ako v predchádzajúcom prípade.

V poslednom prípade je hodnota získavaná pomocou *TypeDescriptor*-u (získavanie hodnoty z jednoduchého objektu). Je potrebné podotknúť, že metóda *GetProperties* vracia len vlastnosti daného objektu a nie členské premenné. V šablónach je teda možné používať len vlastnosti bežných objektov, čo však odpovedá princípom zapuzdrenia a OOP.

Použitie rozhraní a tried zo *System.ComponentModel* namespace predstavuje vysokú abstrakciu prístupu k vlastnostiam dátových zdrojov a umožňuje implementovať široké spektrum rôznych dátových zdrojov pre šablóny. Nespornou výhodou je, že použité prostriedky sú štandardnou súčasťou .NET Frameworku.

4.4 Jazyk pre vyhodnocovanie výrazov

Požiadavok na možnosť podmieneného vykresľovania šablón znamená nutnosť definovať jazyk, pomocou ktorého budú zapisované výrazy, vyžívané značkami *If* a S*witch*. Musí existovať interpret, ktorý tieto výrazy vyhodnotí [4].

Základnou myšlienkou je mať pri generovaní dokumentov možnosť testovať rôzne vlastnosti dátového zdroja, ale aj volať jeho metódy. Užívateľ skonštruuje požadovaný výraz, a na základe jeho hodnoty sa daná značka rozhodne ktorá časť šablóny sa bude vykresľovať. Jazyk by mal byť dynamicky typovaný a interpretovaný, vzhľadom na to že v momente vyhodnocovania nie je známy typ dátového zdroja. Tento jazyk definujeme pomocou bezkontextovej gramatiky.

Bezkontextová gramatiky je štvorica $G=(\prod,\sum,P,S)$ pričom platí:

- $\prod$ je množina neterminálnych symbolov
- $\sum$ je množina terminálnych symbolov, platí $\prod \cap \sum = \emptyset$
- P je počiatočný neterminál
- S je konečná množina prepisovacích pravidiel typu $A \rightarrow \beta$, kde
    - A je neterminál($A \in \prod$)
    - β je reťazec zložený z terminálov a neterminálov, teda $\beta \in (\prod \cup \sum)^*$.

Návrh gramatiky jazyka vychádza zo špecifikácie jazyka C# 3.0, pričom obsahuje len časť definujúcu gramatiku pre výrazy (sú vynechané definície pre riadenia toku programu) [10].

4.4.1 ANTLR

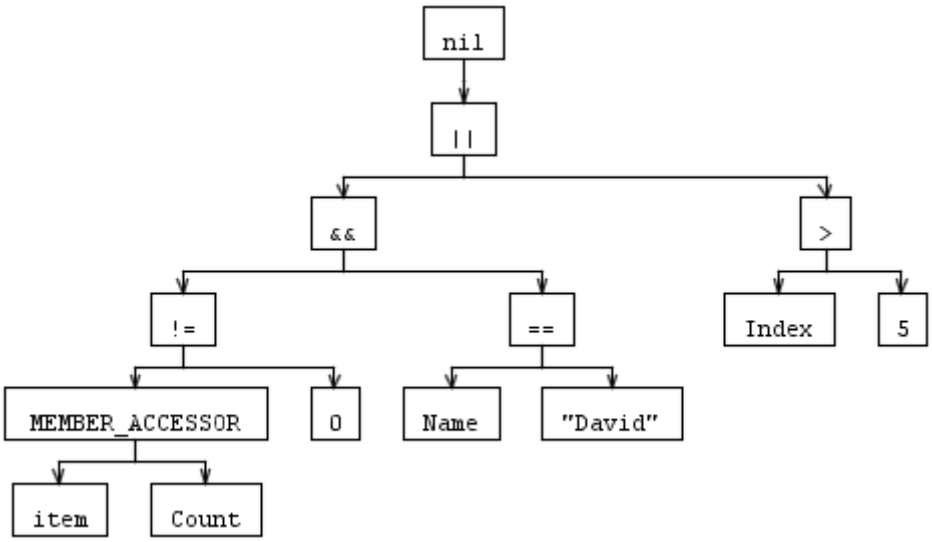
Lexikálna a syntaktická analýza jazyka bola vykonaná pomocou nástroja ANTLR (ANother Tool for Language Recognition). ANTLR automatizuje tvorbu prekladačov, z formálne definovanej gramatiky generuje kód na rozpoznávanie daného jazyka. Výstupom môže byť kód v mnohých

jazykoch, napr. Java, C#, C++, Python atď. K tomuto nástroju existuje vývojové prostredie ANTLRWorks, v ktorom možno jazyky pohodlne vyvíjať a debugovať [2].

Základným vstupom pre ANTLR je formálne definovaná gramatika. Výstupom sú dve vygenerovné triedy, tzv. *Lexer* (lexikálna jazyka) a *Parser* (syntax).

Lexer analyzuje vstupný reťazec a rozdeľuje ho na tzv. lexikálne symboly (tokeny). Lexikálny symbol je charakterizovaný svojou kategóriou (napr. identifikátor, operátor, zátvorka atď.) a atribútmi (kód operátora, hodnota atď.)

Parser slúži na syntaktickú analýzu jazyka a jeho vstupom sú tokeny (vyprodukované *Lexer*-om). Na základe prepisovacích pravidiel danej gramatiky vzniká derivačný strom. ANTLR ponúka široké možnosti na manipuláciu s derivačnými stromami (pridávanie/odoberanie uzlov, definícia nových uzlov, manipuláciu s vetvami a pod.). Po aplikovaní týchto pravidiel vznikne abstraktný syntaktický strom, ktorý je výstupom *parser*-a (Obrázok 10). Slovo abstraktný znamená že strom nereprezentuje každý detail pôvodného derivačného stromu (napr. zátvorky) a jeho štruktúra sa od pôvodného stromu môže líšiť. Strom obsahuje len uzly nutné pre zápis štruktúry výrazu. Tento strom je následne spracovaný interpretom.



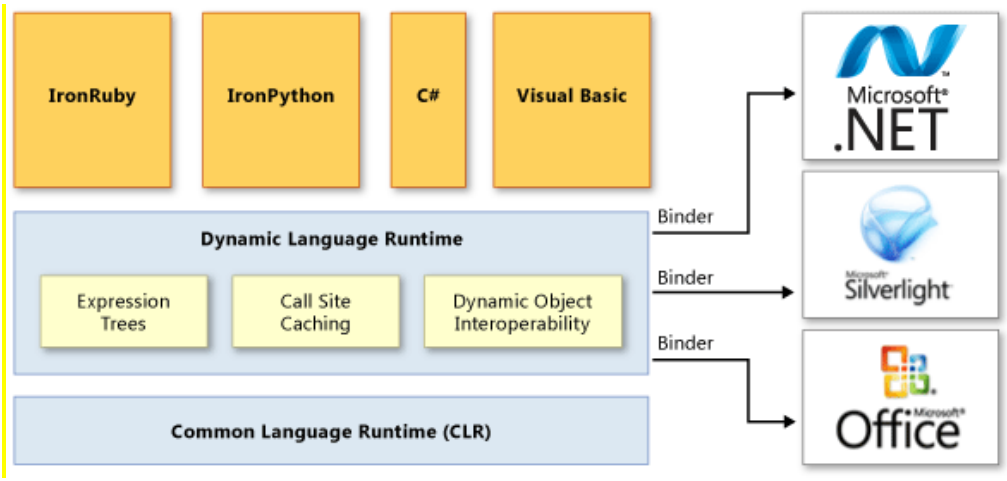
OBRÁZOK 10: ABSTRAKTNÝ SYNTAKTICKÝ STROM

Knižnice ANTLR pre prácu v prostredí .NET Frameworku obsahujú triedy pomocou ktorých je implicitne tvorená výsledná stromová štruktúra, je však možné použiť vlastnú implementáciu stromu.

4.4.2 Dynamic Language Runtime

Dynamic language runtime(DLR) je nadstavba .NET CLR ktorá ho rozširuje o množinu služieb na podporu dynamických jazykov. DLR zjednodušuje vývoj dynamických jazykov bežiacich na platforme .NET a pridáva dynamické prvky do staticky typovaných jazykov.

Dynamické jazyky dokážu určiť typ objektu počas behu programu, zatiaľ čo u staticky typovaných(napr. C#,Java) musí byť typ objektu špecifikovaný počas návrhu programu.



OBRÁZOK 11:ARCHITEKTÚRA DLR [11]

Jednou zo služieb DLR (Obrázok 11) sú takzvané *Expression Trees*. *Expression Trees* reprezentujú sémantiku dynamického jazyka v stromovej štruktúre a poskytujú API pre vytváranie a spúšťanie dynamického kódu. Toto API je rozšírením funkcionality pre dotazovací jazyk *LINQ* a nachádza sa v namespace *System.Linq.Expressions*. Každým uzlom stromu je výraz, napr. volanie metódy alebo relačná operácia.

```
ParameterExpression numParam = Expression.Parameter(typeof(int), "num");
ConstantExpression five = Expression.Constant(5, typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);
Expression<Func<int, bool>> lambda1 = Expression.Lambda<Func<int, bool>>(
                    numLessThanFive,
                    new ParameterExpression[] { numParam });

bool result = lambda1.Compile()(10);
```

VÝPIS 5:TVORBA DYNAMICKÉHO KÓDU POMOCOU EXPRESSION TREES API

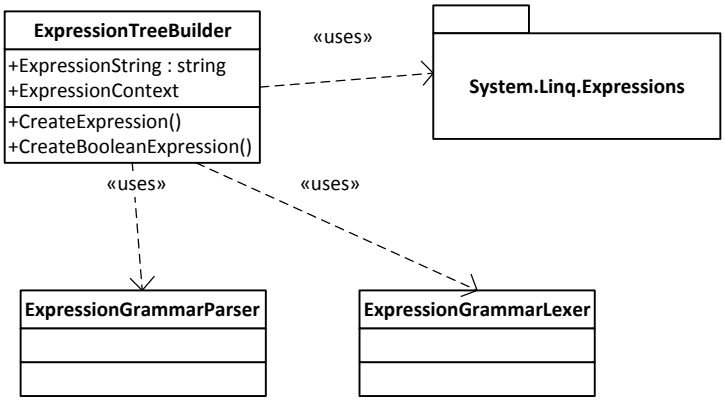
Pomocou *Expression Trees API* je možné jednoducho vytvoriť spustiť dynamický kód (Výpis 5). Táto vlastnosť je kľúčová pre implementáciu interpreta pre náš jazyk. Vzhľadom na to, že definovanie nových typov v jazyku nie je možné, stačí na implementovanie interpreta len *Expression Trees API*.

Keďže výstupom ANTLR *parser*-a z minulej kapitoly je stromová štruktúra, na interpretovanie výrazu stačí previesť vlastný strom na strom reprezentovaný *Expression Trees API*. Túto funkcionalitu implementuje trieda *ExpressionTreeBuilder*.

4.4.3 Trieda ExpressionTreeBuilder

Prevod ANTLR stromu na *Expression Trees API* strom vykonáva trieda *ExpressionTreeBuilder*. Táto trieda má dve vlastnosti:

- *ExpressionString* – reťazec zo vstupným výrazom,
- *ExpressionContext* – predstavuje objekt voči ktorému je výraz vyhodnocovaný.



OBRÁZOK 12:TVORBA AST - ŠTATICKÁ ŠTRUKTÚRA

Obidve metódy triedy *ExpressionTreeBuilder* vracajú generického delegáta typu *Func,* ktorý reprezentuje dynamicky vytvorenú funkciu s návratovou hodnotou rovnakou ako je vstupný výraz. Triedy *ExpressionGrammarLexer* a *ExpressionGrammarParser* sú vygenerované pomocou nástroja *ANTLR*. Celkový algoritmus prevodu vytvorenia ANTLR stromu a jeho prevod na *Expressions Tree API* strom je zobrazený vo výpise č.6.

4.5 Podpisovanie dokumetov

Štandard OPC, na ktorom sú založené OOXML dokumenty, dovoľuje aby boli balíčky podpísané pomocou digitálneho podpisu. Podpisovanie OPC balíčkov je založené na štandarde XML Digital Signature Standard (XML-DSIG) [15], ktorý je definovaný v doporučení W3C XML-Signature Syntax and Processing. Súčasťou implementácia OPC v .NET Frameworku je OPC Digital Signing Framework, ktorý implementuje podpisovanie OPC balíčkov pomocou X509 certifikátov.

Každý OPC balíček sa skladá z častí a väzieb (Kapitola 4.1). Pre každý formát, ktorý je založený na OPC je definovaná podpisová politika, ktorá určuje ako podpisovať a overovať obsah dokumentu. Táto politika definuje, ktoré časti a väzby podpisovať a ktoré zostanú nepodpísané. Pre konkrétny formát môže počas životného cyklu dokumentu existovať viacero podpisových politík.

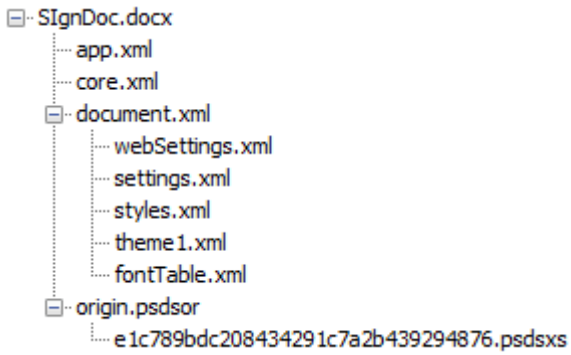
```
//Parsovanie výrazov pomocou tried vygnerovaných nástrojom ANTLR
ANTLRStringStream input = new ANTLRStringStream(ExpressionString);
ExpressionGrammarLexer lexer = new ExpressionGrammarLexer(input);
CommonTokenStream tStream = new CommonTokenStream(lexer);
ExpressionGrammarParser parser = new ExpressionGrammarParser(tStream);
ExpressionGrammarParser.expression_return result = parser.expression();

//CommnonTree je AST vytvorený pomocou ANTLR
CommonTree tree = result.Tree as CommonTree;

//Metóda BuildTreeRecursive prevádza ANTLR AST na Expressions Tree API AST
Expression expression = BuildTreeRecursive(tree);
Expression<Func<bool>> lambda = null;
try
{
	//Vytvorenie lambda výrazu
	lambda = Expression.Lambda<Func<bool>>(expression);
}
//Skompilovaním lambdy vznikne delegát typ Func
return lambda.Compile();
```

VÝPIS 6:PREKLAD VÝRAZU NA DELEGÁTA FUNC

Pri podpise dokumentu vznikajú tzv. podpisová časť (signature part),ktorá obsahuje informácie o podpísaných väzbách a častiach. Certifikát použitý k podpisu môže byť súčasťou tejto časti, alebo môže tvoriť samostatnú časť balíčka. Tieto podpisové časti sú odkazované zo špeciálnej koreňovej časti nazvanej *digital signature origin*.



OBRÁZOK 13: ŠTRUKTÚRA PODPÍSANÉHO DOKUMENTU

Obrázok 13 ukazuje štruktúru podpísaného dokumentu. Časť *origin.psdsor* je *digital signature origin* časť, ktorá sa odkazuje na samotnú podpisovú časť. Podpisová časť je vlastne XML súbor, ktorého obsah je definovaný štandardom W3C XML-DSIG. Elementom v rámci ktoré sú uchovávané informácie o samotnom podpise je element *Object* (Výpis 7).

```
<Object Id="idPackageObject">
<Manifest  xmlns:opc="http://schemas.openxmlformats.org/package/2006/digital-
signature">
  <Reference
     URI="/word/document.xml?ContentType=application/vnd.openxmlformats-
     officedocument.wordprocessingml.document.main+xml">
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
       <DigestValue>81OYDEMbUEggEjdz4IcFvMVKRX0=</DigestValue>
  </Reference>
   .
   .
   .
</Object>
```

VÝPIS 7: OBSAH ELEMENTU OBJECT

Obsah elementu *Object* nie je štandardom W3C nijako definovaný, je v ňom možné uchovávať rôzne informácie o digitálnom podpise. OPC v tomto elemente definuje elementy, nesúce informácie o podpise jednotlivých častí a väzieb. Koreňovým elementom je *Manifest* ktorý obsahuje viacero elementov *Reference*.

Element *Reference* obsahuje odkaz na konkrétnu časť balíčka a informácie o použitom hashovaciom algoritme a hodnote odtlačku pre danú časť. Je dôležité podotknúť, že pred výpočtom odtlačku XML súboru je potrebné vykonať jeho kanonizáciu algoritmom určeným v štandarde W3C, z dôvodu aby generovanie odtlačku pri podpisovaní a overovaní podpisu pracovalo nad rovnaký XML reťazcom. Informácia o použitom kanonizačnom algoritme je súčasťou danej podpisovej časti.

4.6 Značky

Samotné spracovanie šablóny bolo špecifikované sekvenčným diagramom (Obrázok 4). Inštancia triedy *WordParser* teda prechádza rekurzívne šablónu a pri customXML elementu reprezentujúceho značku, vytvorí inštanciu triedy pre danú značku ktorej predá ďalšie spracovanie.

Bázovou triedou pre značky je abstraktná trieda *TemplateTag*. Táto trieda drží referencie na dôležité objekty ako sú spracovávaný customXML element a inštancia *WordParser*-a. Všetky triedy značiek musia svoju logiku implementovať prostredníctvom abstraktnej metódy *ProcessElement*.

Pri spracovaní šablóny štandardne dochádza k odstráneniu customXML značiek. Podrobný popis jednotlivých značiek je v prílohe C.

Chybové stavy počas generovania dokumentu sú reprezentované triedou *ParserException*. Táto trieda predstavuje výnimku ktorá je štandardne vyhadzovaná v prípade chyby, tak aby užívateľ integrujúci komponentu mohol na ňu adekvátne reagovať. *ParserException* nesie krátky popis chyby a prípadnú vnútornú výnimku (inner exception) ak nastala. V prípade chýb v šablóne je k

dispozícii text dokumentu v okolí kde nastala chyba, čo umožňuje jednoducho lokalizovať chybu v šablóne. Toto je obzvlášť užitočné pri tvorbe a ladení šablóny.

4.7 Rozšíriteľnosť

Komponenta bola navrhnutá s ohľadom na možné rozšírenie funkcionality v budúcnosti. Základnou myšlienkou je umožniť rozširovať funkcionalitu komponenty pomocou nových značiek. K tomuto účelu bola definovaná značka *customtag*.

Táto značka má jeden atribút *type*, ktorý obsahuje tzv. *assembly qualified name*, čo je meno typu ktorý implementuje rozhranie *IOpenXmlCustomProcessor.* Toto rozhranie obsahuje jedinú metódu *ProcessElement*, ktorá ako argumenty berie aktuálny Open XML element a dátový zdroj. Pomocou tohto rozhrania teda možno implementovať ľubovoľnú užívateľskú funkcionalitu.

Je dôležité podotknúť že assembly s typom implementujúcim spomínané rozhranie musí byť načítaná v behovom prostredí a aplikačnom poole, v ktorom beží komponenta.

4.8 Trieda OpenXmlUtils

Na základe požiadavku na vkladanie textových polí, obrázkov a podpisovania dokumentov bola vytvorená trieda *OpenXmlUtils.* Táto trieda obsahuje statické metódy na manipuláciu s vygenerovanými dokumentmi.

Podpisovanie dokumentov bola rozobraté v kapitole 4.5, preto sa budem ďalej zaoberať len vkladaním textových polí a obrázkov. Obrázky a textové polia je možné vkladať pomocou metód *InsertImageIntoDocument,* resp. *InsertTextBox.* Pre vloženie elementov je nutné špecifikovať miesto v dokumentu. V prípade vloženia do prvého odstavca (čo je štandardné správanie pri volaní týchto metód) sa daný elementu bude nachádzať vždy na prvej strane dokumentu. Toto správanie vyplýva z návrhu technológie OOXML.

Pre prípad potreby vložiť element na iné miesto v dokumente, napr. na koniec dokumentu, alebo do tabuľky, bol špecifikovaný nový customXML element *placeholder*. Tento element je identifikovaný svojim menom (atribút *name*) a je potrebné ho vopred vložiť do dokumentu do ktorého chceme vkladať obrázok alebo textové pole (môže byť však súčasťou samotnej šablóny, komponenta tieto značky pri generovaní ignoruje, čiže sú súčasťou vygenerovaných dokumentov). Pri samotnom vkladaní sa odkazujeme na tento element pomocou jeho názvu (je to jeden z parametrov metód).

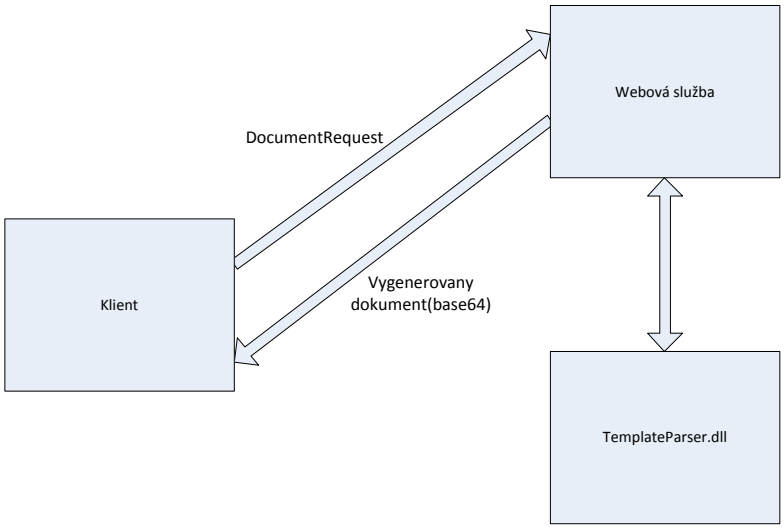
5 Implementácia

Riešenie bolo implementované na platforme .NET Framework v.4, v jazyku C#. Ako vývojové prostredie bolo použité MS Visual Studio 2010, na vývoj a ladenie gramatiky jazyka pre vyhodnocovanie výrazov zasa ANTLRWorks 1.4. Na manipuláciu s Word dokumentmi bolo použité MS Open XML SDK.

5.1 Webová služba – reportovací server

Súčasťou implementácie bolo vytvorenie webovej služby, ktorá integrovala hotovú komponentu. Výsledkom bol jednoduchý reportovací server, ktorý na základe požiadavku vráti vygenerovaný dokument.

Rozhranie webovej služby je definované pomocou protokolu SOAP. Klient posiela metóde webovej služby objekt *DocumentRequest* na základe ktorého webová služba vracia požadovaný dokument zakódovaný vo formáte base64 (Obrázok 14).



OBRÁZOK 14: SCHÉMA WEBOVEJ SLUŽBY

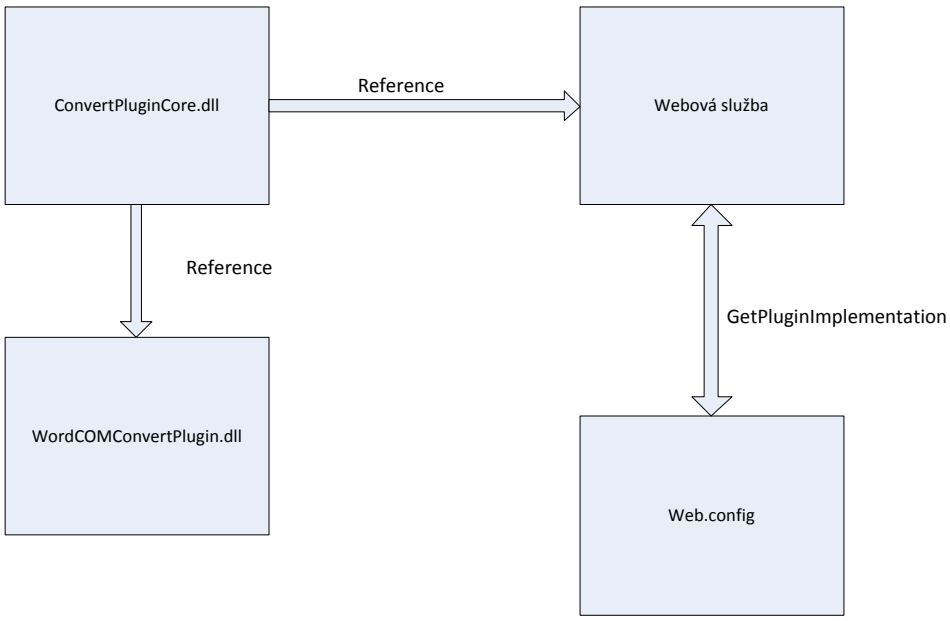
Objekt *DocumentRequest* obsahuje nasledujúce elementy:

- *TemplateName* – meno šablóny, označenie pod ktorým je šablóna uložená na reportovacom serveri,
- *Template* – v prípade že šablóna nie je uložená na serveri, je možné ju poslať zakódovanú v base64 v požiadavku,
- *DataSourceType* – typ objektu dátového zdroja, musí byť známy kvôli deserializácii,
- *DataSourceXML* – objekt dátového zdroja, serializovaný do XML reťazca,
- *OutputFormat* – požadovaný výstupný formát (Doc, Docx, Pdf, Rtf).

5.2 Konverzia na iné formáty

Konverzia na iné výstupné formáty nie je súčasťou implementácie komponenty, preto bol navrhnutý pluginový systém, ktorý umožní k webovej službe pridávať pluginy na konverziu dokumentov do rôznych výstupný formátov. Implementovaný bol plugin na konverziu do formátu Word 2003, ktorý dokumenty konvertuje pomocou automatizácie COM objektu aplikácie MS Word 2007.

Základom pluginového systému je rozhranie *IConvertPlugin*, ktoré je publikované v assembly *ConvertPluginCore.dll*. Implementovať plugin znamená implementovať toto rozhranie a zverejniť ho ako samostatnú assembly (.dll súbor). V konfigurácii webovej služby stačí uviesť že konkrétny plugin je implementovaný v danom .dll súbore.



OBRÁZOK 15:PLUGINOVÝ SYSTÉM

5.3 Výkonnostné testy

Výsledná implementácia novej komponenty a takisto starý systém boli podrobené výkonnostným testom, na základe ktorých je možné určiť výkonnostný rozdiel medzi oboma riešeniami. Záťažové testy boli pomocou nástrojov ktoré ponúka vývojové prostredie Visual Studio 2010 Ultimate. Táto verzia umožňuje automatizované testovanie záťaže aplikácii, vrátane simulovania mnohoužívateľského prístupu k aplikácii.

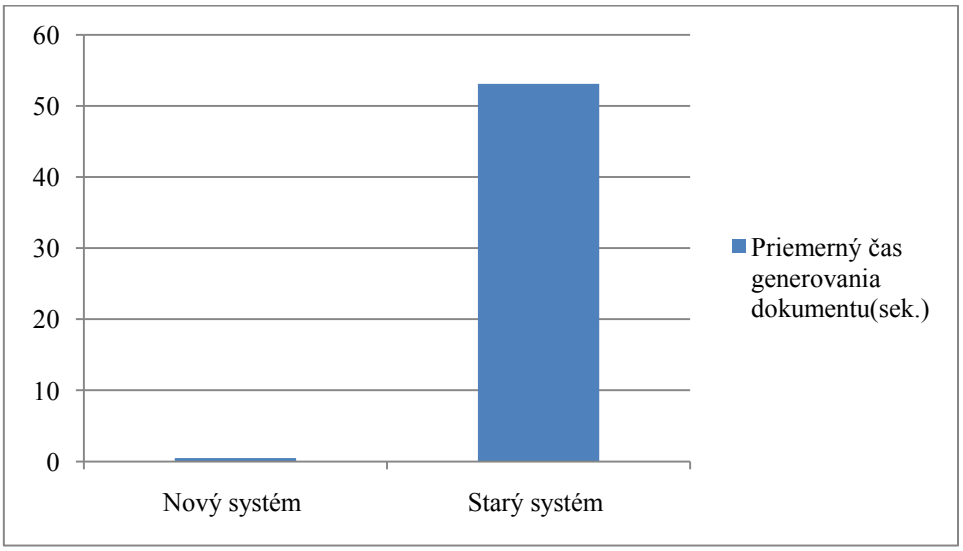
Základom pre porovnanie výkonnosti oboch komponent bolo vytvoriť testovacia šablóny s približne rovnakou funkcionalitou. Pre každú z komponent bola vytvorená jedna testovacia šablóna, tak aby boli výstupné dokumenty čo najviac podobné.

Pre každú z komponent bola vytvorená samostatná webová služba, ktorá generovala testovacie dokumenty. V teste nebol zahrnutý prístup k dátam (čo ani nebolo účelom testu), dátové zdroje - objekty boli vopred (staticky) nadefinované.

Na webové služby bol simulovaný mnohoužívateľský prístup, pričom bol meraný priemerný čas spracovania požiadavku na dokument. Počet súčasných užívateľov bol na počiatku nastavený na 2, pričom sa zvyšoval postupne na 20 (každých 5 sekúnd o 2 užívateľov) za účelom sledovať reakciu aplikácie. Test každej webovej služby trval približne dve minúty. Testovacie webové služby boli nasadené na virtuálnom serveri s konfiguráciou Intel Xeon 3.4 GHz a 2 GB RAM.

	Max. počet užívateľov	Počet požiadavkov/sekunda	Priemerný čas generovania dokumentu(sek.)
Nová komponenta	20	33,9	0,48
Stará komponenta	20	0,11	53,1

TABUĽKA 4:POROVNANIE VÝKONU KOMPONENT

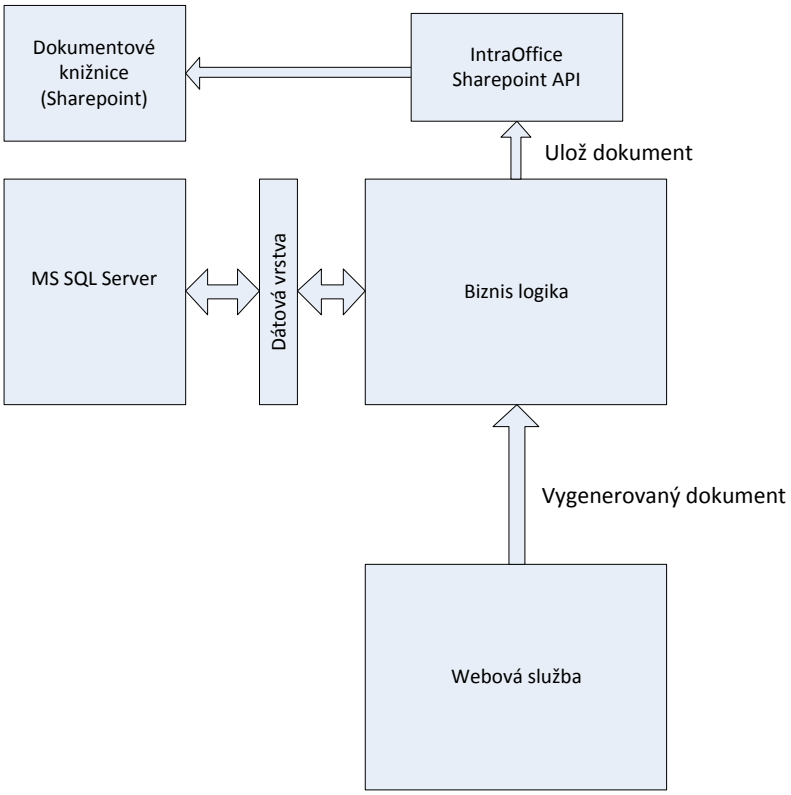


OBRÁZOK 16:PRIEMERNÝ ČAS GENEROVANIA DOKUMENTU

Na základe tabuľky a grafov môžeme vidieť výrazný rozdiel vo výkone oboch komponent. Výrazný náskok nového riešenia má na svedomí najmä fakt že dokument je reprezentovaný sadou XML súborov. Spracovanie textu (XML) je, ako vidieť, rýchlejšie ako automatizovanie COM komponenty Wordu, ktoré si vyžaduje značné pamäťové a výpočtové prostriedky.

6 Nasadenie riešenia v praxi

Implementované riešenie bolo integrované do informačného systému IntraOffice firmy ELINKX. Tento webový informačný systém je nadstavbou pre MS Sharepoint Server a implementuje široké spektrum funkcionality v oblasti správy dokumentov a podnikového workflow. V rámci jednotlivých modulov je možné generovať rôzne druhy reportov, ktoré sú ukladané do dokumentových knižníc Sharepoint-u. Ako dátovu vrstvu využíva systém MS SQL Server.



OBRÁZOK 17:INTEGRÁCIA DO INTRAOFFICE

V rámci integrácie bolo nutné vytvoriť biznis objekty, na ktoré budú namapované relačné dáta z databázy MS SQL Server (informačný systém využíva vlastný framework na objektovo-relačné mapovanie). Tieto biznis objekty tvoria dátové zdroje pre report. Nutnú úpravu si vyžadovala aj biznis logika, pretože generovanie dokumentov sa presunulo na webovú službu, ktorá je popísaná v kapitole 5.1. Presunúť generovanie dokumentov na samostatný server prináša výhodou v jednoduchej škálovateľnosti. V prípade nedostatočného výkonu možno ľahko nahradiť clusterom, prípadne cluster rozšíriť o ďalší server.

Úložiskom pre vygenerované dokumenty sú dokumentové knižnice Sharepoint-u, do ktorých sú dokumenty ukladané pomocou proprietárneho API firmy E LINKX.

V budúcnosti sa plánuje integrácia komponenty do informačného systému, ktorý využíva Nejvyšší správní soud (NSS) ČR, kde má nahradiť starý systém generovania dokumentov. Prostredníctvom tejto komponenty komponenty teda budú generované všetky dokumenty NSS.

7 Záver

Obsahom tejto práce bola problematika generovania Word dokumentov. Na začiatku bolo staré riešenie, ktoré bolo potrebné inovovať. Po zhodnotení analýzy výhod a nevýhod tohto riešenia, bolo navrhnuté riešenie nové, založené na súčasných technológiách ako je formát Office Open XML, či .NET Framework 4.

Nová komponenta prináša celu radu výhod, ako je napr. jednoduchý dizajn šablón prostredníctvom aplikácie MS Word 2007, rozšírenie funkčnosti o vkladanie obrázkov, či iných dokumentov alebo možnosť definovať prezentačnú logiku priamo v šablóne pomocou vlastného jazyka pre tvorbu výrazov, pričom je jednoduché ďalšie rozšírenie funkčnosti pomocou užívateľských značiek.

Súčasne bol implementovaný aj jednoduchý reportovací server založený na webovej službe a celé riešenie bolo integrované to prakticky používanej aplikácie. V ďalšej fáze sa predpokladá integrovanie komponenty do informačného systému Nejvyššícho správního soudu ČR.

Možné rozšírenie do budúcnosti predstavuje implementácia pluginov na prevod do rôznych formátov. Zároveň je komponenta navrhnutá tak, aby sa dala rozšíriť aj o spracovanie nových typov šablón, napr. vo formáte HTML. Na základe patentového sporu Microsoftu ohľadom technológie CustomXML by sa mal v ďalšom vývoji zvážiť návrh nového značkovacieho jazyka, prípadne aplikácie dizajnéra šablón.

8 Referencie

[1] REDMOND, Frank E. *Dcom: Microsoft Distributed Component Object Model*. [s.l.] : John Wiley & Sons Inc, 1997. 359 s. ISBN 0764580442.

[2] PARR, Terrence. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. [s.l.] : Pragmatic Bookshelf, 2007. 376 s. ISBN 0978739256.

[3] VUGT, Wouter van. *Open XML - The markup explained* [online]. [s.l.] : [s.n.], 2007 [cit. 2010-05-13]. Dostupné z WWW: < http://openxmldeveloper.org/attachment/1970.ashx >.

[4] BENEŠ, Miroslav. *Překladače* [online]. Ostrava : VŠB-TUO, [200?]. 123 s. Skriptum. VŠB-TUO Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky. Dostupné z WWW: < http://www.cs.vsb.cz/behalek/vyuka/pjp/skripta/skr-mb.pdf >.

[5] E LINKX a.s. *E LINKX a.s.* [online]. 1999 [cit. 2011-05-01]. Dostupné z WWW: < http://www.elinkx.cz >.

[6] SoftArtisans. *SoftArtisans* [online]. 2011 [cit. 2011-05-01]. Dostupné z WWW: < http://www.softartisans.com/ >.

[7] Ecma international. *Ecma international* [online]. 2006 [cit. 2010-06-10]. ECMA-376 1st edition Part 2. Dostupné z WWW: < http://www.ecma-international.org/publications/files/ECMA-ST/Office%20Open%20XML%201st%20edition%20Part%202%20(PDF).zip >.

[8] Ecma international. *Ecma international* [online]. 2006 [cit. 2010-08-16]. ECMA-376 1st edition Part 4. Dostupné z WWW: < http://www.ecma-international.org/publications/files/ECMA-ST/Office%20Open%20XML%201st%20edition%20Part%204%20(PDF).zip >.

[9] Ecma international. *Ecma international* [online]. 2006 [cit. 2010-06-13]. OpenXML White Paper. Dostupné z WWW: < http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf >.

[10] Microsoft. *Microsoft Developer Network* [online]. 2006 [cit. 2011-04-04]. C# Language Specification - C. Grammar. Dostupné z WWW: < http://msdn.microsoft.com/en-us/library/aa664812(v=vs.71).aspx >.

[11] Microsoft. *Microsoft Developer Network* [online]. 2009 [cit. 2011-03-10]. Dynamic Language Runtime Overview. Dostupné z WWW: < http://msdn.microsoft.com/en-us/library/dd233052.aspx >.

[12] Microsoft. *Microsoft Support* [online]. 2010 [cit. 2011-05-01]. Description of the January 2010 update for Word 2003 and for Word 2007. Dostupné z WWW:
< http://support.microsoft.com/kb/978951 >.

[13] Microsoft. *MS Sql Server 2008* [online]. 2008 [cit. 2011-04-28]. Reporting Services. Dostupné z WWW: < http://www.microsoft.com/sqlserver/2008/en/us/reporting.aspx >.

[14] The Unicode Consortium. *The Unicode Consortium* [online]. 2011 [cit. 2011-05-01]. The Unicode Standard. Dostupné z WWW: < http://www.unicode.org/versions/latest/ >.

[15] W3C. *W3C* [online]. 2008 [cit. 2011-05-01]. World Wide Web Consortium. Dostupné z WWW: < http://www.w3.org/Signature/ >.

A XML schéma značiek

XML schéma obsahuje definíciu XML elementov, ktoré reprezentujúc jednotlivé značky šablóny. Pomocou tejto schémy sa "importujú" značky do aplikácie MS Word.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="WordDocumentTemplateSchema"
    targetNamespace="http://tempuri.org/XMLSchema1.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema1.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <xs:element name="property">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="formatString" type="xs:string" use="optional" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="foreach">
    <xs:complexType mixed="true">
      <xs:group ref="statement"></xs:group>
      <xs:attribute name="in" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="include">
    <xs:complexType>
        <xs:attribute name="src" type="xs:string" use="optional" />
        <xs:attribute name="file" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="if">
    <xs:complexType mixed="true">
      <xs:all>
        <xs:element ref="then" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="else" minOccurs="0" maxOccurs="1"/>
      </xs:all>
      <xs:attribute name="expression" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="then">
    <xs:complexType mixed="true">
        <xs:group ref="statement" />
    </xs:complexType>
  </xs:element>

  <xs:element name="else">
    <xs:complexType mixed="true">
      <xs:group ref="statement"></xs:group>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name="switch">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="case" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="default" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="expression" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="case">
    <xs:complexType mixed="true">
      <xs:group ref="statement"></xs:group>
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="default">
    <xs:complexType mixed="true">
      <xs:group ref="statement"></xs:group>
    </xs:complexType>
  </xs:element>

  <xs:element name="inlineimage">
    <xs:complexType mixed="true">
      <xs:attribute name="path" type="xs:string" use="optional" />
      <xs:attribute name="width" type="xs:double" use="optional" />
      <xs:attribute name="height" type="xs:double" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="anchorimage">
    <xs:complexType mixed="true">
      <xs:attribute name="path" type="xs:string" use="optional" />
      <xs:attribute name="width" type="xs:double" use="optional" />
      <xs:attribute name="height" type="xs:double" use="optional" />
      <xs:attribute name="left" type="xs:integer" use="optional" />
      <xs:attribute name="top" type="xs:integer" use="optional" />
      <xs:attribute name="v-align" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="center"/>
            <xs:enumeration value="inside"/>
            <xs:enumeration value="top"/>
            <xs:enumeration value="outside"/>
            <xs:enumeration value="bottom"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="h-align" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="center"/>
            <xs:enumeration value="inside"/>
```

```
            <xs:enumeration value="left"/>
            <xs:enumeration value="outside"/>
            <xs:enumeration value="right"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="text-wrap" use="optional" >
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="none"/>
            <xs:enumeration value="square"/>
            <xs:enumeration value="topbottom"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="v-pos-relative-from" use="optional" >
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="topmargin"/>
            <xs:enumeration value="bottommargin"/>
            <xs:enumeration value="page"/>
            <xs:enumeration value="insidemargin"/>
            <xs:enumeration value="outsidemargin"/>
            <xs:enumeration value="margin"/>
            <xs:enumeration value="line"/>
            <xs:enumeration value="paragraph"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="h-pos-relative-from" use="optional" >
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="leftmargin"/>
            <xs:enumeration value="rightmargin"/>
            <xs:enumeration value="page"/>
            <xs:enumeration value="column"/>
            <xs:enumeration value="character"/>
            <xs:enumeration value="insidemargin"/>
            <xs:enumeration value="outsidemargin"/>
            <xs:enumeration value="margin"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name="placeholder">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="customtag">
    <xs:complexType>
      <xs:attribute name="type" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
```

```
  <xs:group name="statement">
    <xs:choice>
      <xs:element ref="property" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="foreach" minOccurs="0" maxOccurs="unbounded" />
      <xs:element ref="if" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="include" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="switch" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="anchorimage" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="inlineimage" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="placeholder" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="customtag" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:group>
</xs:schema>
```

VÝPIS 8:XML SCHÉMA ZNAČIEK V DOKUMENTE

B Jazyk pre vyhodnocovanie výrazov

V tejto prílohe sú popísane jednotlivé časti jazyka. Jazyk vychádza zo špecifikácie výrazov jazyka C#, pričom vynecháva niektoré operátory a konštrukcie. Výrazy jazyk sa skladajú z dvoch prvkov – operátory a operandy. Operandy sú tvorené identifikátormi alebo literálmi.

B.1 Identifikátory

Tvar identifikátorov je totožný s definíciou identifikátorov v jazyku C#. Identifikátor musí začínať písmenom, prípadne znakom "_" (podtržítko). Ďalej môžu nasledovať ľubovoľnom poradí čísla, písmená a znaky patriace do tried Mn,Mc,Nd,Pc,Cf Unicode štandardu [14].

Jazyk pracuje nad dátovým zdrojom, preto jednoduché identifikátory reprezentujú členov objektu (vlastnosti, členské premenné, metódy). Samozrejme, keďže členovia objektu majú taktiež svojich členov, je možné k nim pristupovať pomocou bodky (napr. prístup k počtu členov kolekcie – *item.Count*, kde *item* je názov člena dátového zdroja, ktorý ma vlastnosť *Count*). Obdobne je možné volať metódy objektov. V prípade že volanie metódy je uvedené bez identifikátora objektu (napr. vo výraz *GetName() == "name"*) je príslušná metóda braná ako inštančná metóda objektu dátového zdroja.

Jazyk nepodporuje prístup k členom predefinovaných typov ako sú napr. *bool, double* atď. (napr. nieje možný prístup *int.MinValue* a pod) a takisto nie je možné volať statické metódy ani pristupovať k ostatným statickým členom objektov.

B.2 Literály

Literály opäť vychádzajú z jazyka C# a môžeme ich rozdeliť na:

- Booleovské – reprezentované reťazcami *true* alebo *false,*
- Celočíselné – reprezentované pomocou desiatkových čísel, prípadne pomocou hexadecimálneho zápisu (s prefixom *0x*). Každý celočíselný literál môže mať sufix, ktorý určuje je typ (*U, UL, L,LU*). Pred zápis literálu je možné uviesť znamienko (+,-),
- Reálne – reprezentujú reálne čísla, desatinný oddeľovač je znak "." (bodka). Za zápisom čísla môže nasledovať časť s exponentom označená písmenom *E* a príslušnou hodnotou exponentu (napr. 10.45E22). Každý reálny literál môže mať sufix, ktorý určuje jeho dátový typ. (*F,D,M*). Pred zápis literálu je možné uviesť znamienko (+,-),
- Znakové – reprezentuje Unicode znak, musí byť uzatvorený pomocou znakov ' (napr. *'a'*). Znak je možné reprezentovať aj pomocou *escape* sekvencie, alebo pomocou hexadecimálneho kódu znaku (s prefixom *\x*),
- Reťazcové – reprezentujú reťazec znakov, musí byť uzatvorený pomocou znakov " (napr. *"hello World"*). V prípade že sa pred literál uvedie znak @ ignorujú sa prípadne špeciálne znaky v reťazci, t.j. reťazec sa berie ako doslovný,
- Null literál – reprezentovaný reťazcom *null*.

B.3 Operátori

V nasledujúcej tabuľke sú uvedené operátori jazyka, zoradené podľa ich priority (operátori rovnakej kategórie majú rovnakú prioritu). Prioritu operátorov možno meniť pomocou jednoduchých zátvoriek.

Kategória	Výraz	Popis
Primárne	`x.m`	Prístup k členovi
	`x(...)`	Volanie metódy alebo delegáta
	`x[...]`	Prístup k poľu alebo indexeru
	`x++`	Postfixový inkrement
	`x--`	Postfixový dekrement
Unárne	`+x`	Identita
	`-x`	Negácia
	`!x`	Logická negácia
Multiplikatívne	`x * y`	Násobenie
	`x / y`	Delenie
	`x % y`	Zvyšok po delení
Aditívne	`x + y`	Sčítanie
	`x - y`	Odčítanie
Relačné	`x < y`	Menši než
	`x > y`	Väčší než
	`x <= y`	Menší alebo rovný než
	`x >= y`	Väčší alebo rovný než
Rovnosť	`x == y`	Rovnosť
	`x != y`	Nerovnosť
Podmienený AND	`x && y`	Vyhodnocuje y len ak x je pravda
Podmienený OR	`x || y`	Vyhodnocuje y len ak x je nepravda

TABUĽKA 5:OPERÁTORY JAZYKA

V prípade že možnosti jazyk nepostačujú potrebám, je vhodnou stratégiou implementovať požadovanú funkcionalitu ako metódu objektu dátového zdroja a tú volať prostredníctvom výrazu.

Na nasledujúcich príkladoch sú ilustrované možnosti jazyka:

Pr.1: *Customers.Count!= 0 || IsValid()*

Výraz testuje či vlastnosť *Customer* má nenulový počet prvkov alebo metóda *IsValid* vracia *true*.

Pr.2 *Invoice.Items > 100 && (Customer.IsReseller() || Name != "Ostrava")*

Na tomto príklade je ilustrovaná zmena priority operátorov a ukážka použitia relačných operátorov.

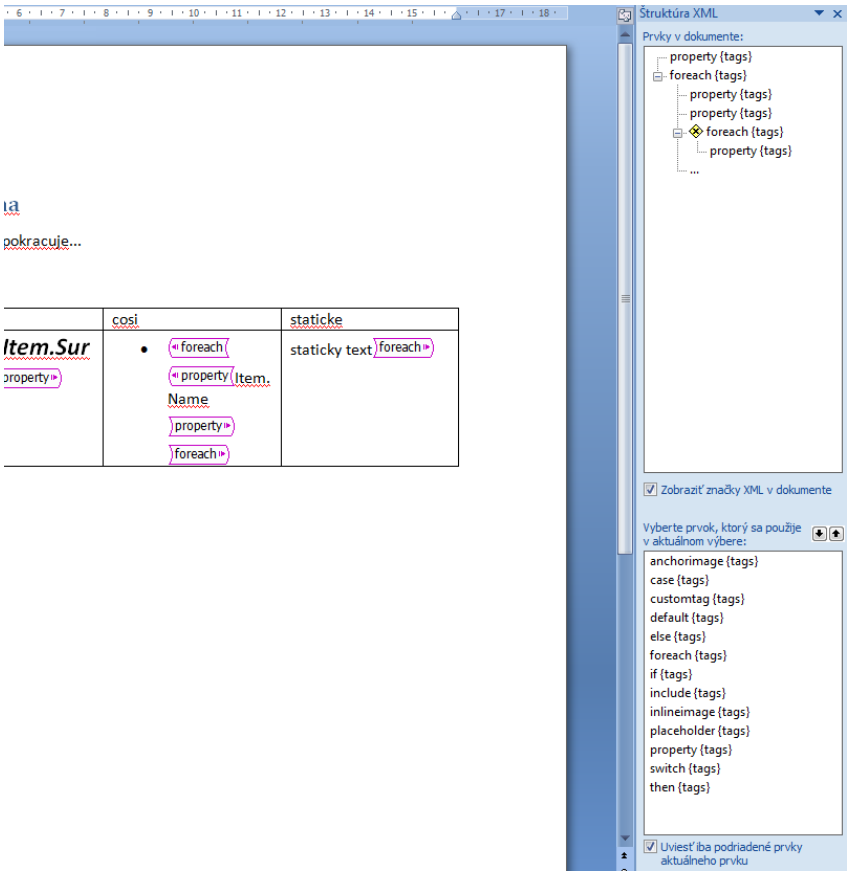
C Používateľská dokumentácia

C.1 Tvorba šablón pomocou MS Word 2007

MS Word 2007 podporuje vkladanie customXML značiek do dokumentu, na základe ktorých funguje aj komponenta generovania dokumentov. Aby sme mohli so vlastnými značkami pracovať, je potrené pridať XML schému s ich definíciou medzi schémy používané aplikáciou MS Word.

Postup pri pridávaní schémy je nasledovný(názvy sú vzťahované k slovenskej verzii MS Word):

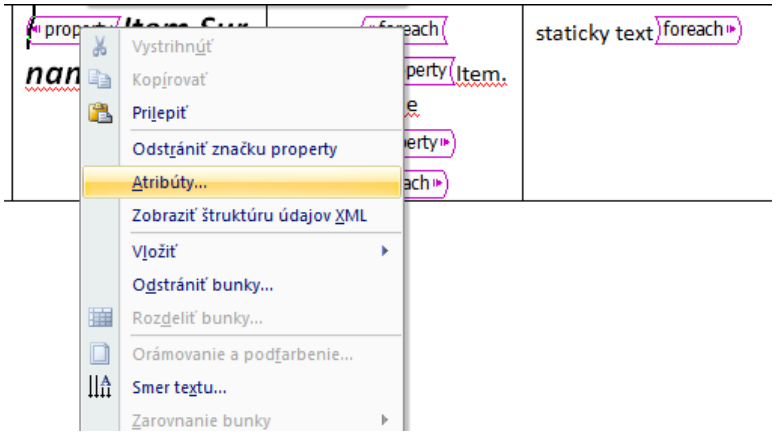
1. Zapnúť kartu *Vývojár* – v kliknutím na ikonku loga MS Office v pravom hornom rohu, v spodnej časti vyberieme položku *Možnosti programu Word*. V sekcii *Obľúbené* zaškrtneme *Zobraziť kartu vývojár na páse s nástrojmi.* V menu pribudne spomínaná karta.
2. Na karte *Vývojár* klikneme na tlačidlo *Schéma*. Na karte *Schéma XML* klikneme na tlačidlo *Pridať schému*.
3. Pridanú schému pomenuje a pridáme zaškrtnutím.



OBRÁZOK 18:TVORBA ŠABLÓN V MS WORD 2007

Po pridaní schémy sa zobrazí napravo panel *Štruktúra XML* (Obrázok 18) s dostupnými značkami. V prípade že sa tak nestane, zobrazíme ho pomocou tlačidla *Štruktúra* na karte *Vývojár*.

Pridávaním značiek z panelu do dokumentu tvoríme šablónu. V prípade že chceme danej značke nastaviť atribúty, klikneme na ňu pravým tlačidlom a z kontextového menu vyberieme možnosť *Atribúty*.



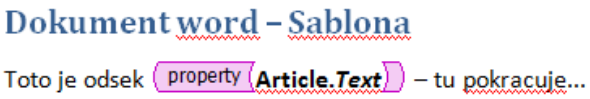
OBRÁZOK 19:NASTAVENIE ATRIBÚTOV ZNAČKY

C.2 Značky

V tejto časti budú jednotlivé značky rozobraté z pohľadu tvorby šablón. Obsah značiek *if, switch, foreach* sa dá ľubovoľne zanorovať, t.j. môžu obsahovať opäť ďalšie značky.

Property

Táto značka je pri spracovaní nahradená hodnotou príslušnej vlastnosti dátového zdroja. Meno vlastnosti a štýl sú určené textom vnútri značky. Značke je možné nastaviť pomocou atribútu *formatString* formátovací reťazec v tvare *{0:str}*, reťazec *str* predstavuje štandardný .NET formátovací reťazec.

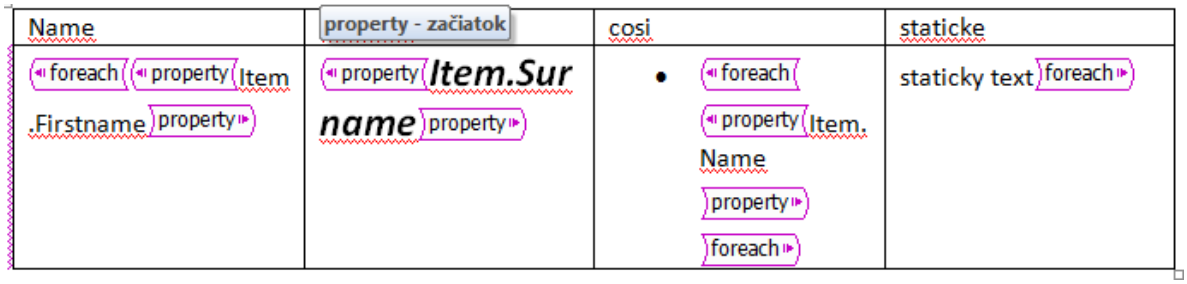


OBRÁZOK 20:ZNAČKA PROPERTY

Foreach

Táto značka iteruje cez kolekciu zadanú ako atribút *in* a pre každý prvok spracováva časť dokumentu označenú touto značkou. Označením blokových častí dokumentu(odseky, riadky tabuľky, celé tabuľky) je možné generovať sadu odsekov, riadkov v tabuľkách atď. Dátovým zdrojom pre značky vo vnútri tejto značky je špeciálny kontext objekt, ktorý obsahuje informácie o

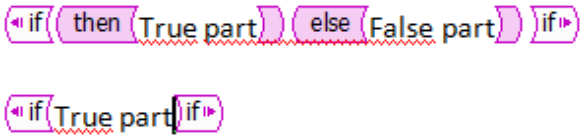
celkovom počte objektov v kolekcii(*Length*) a aktuálnom indexe(*CurrentIndex*). K pôvodnému elemetu kolekcie sa dá dostať pomocou vlastnosť *Item*.



OBRÁZOK 21:ZNAČKA FOREACH

If

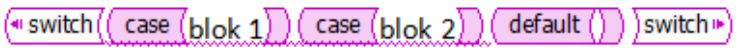
Prostredníctvom tejto značky je možné podmienene vykresľovať časti šablóny. Podmienka je vyjadrená výrazom v atribúte *expression*. Vo vnútri značky musia byť použité buď obidva detské elementy *then* a *else*, alebo ani jeden z nich. V prvom prípade sa na základe platnosti podmienky vykreslí buď časť *then* alebo *else*, v druhom sa vykreslí obsah značky len ak je podmienka platná.



OBRÁZOK 22:ZNAČKA IF

Switch

Táto značka umožňuje zvoliť na základe viacerých možných hodnôt výrazu ktorá časť sa bude vykresľovať. Výraz je definovaný prostredníctvom atribútu *expression*. Detské elementy sú *case* a *default*. Element *case* má atribút *value* kde je definovaná konkrétna hodnota výrazu pre daný element. Element *default* predstavuje predvolený blok.



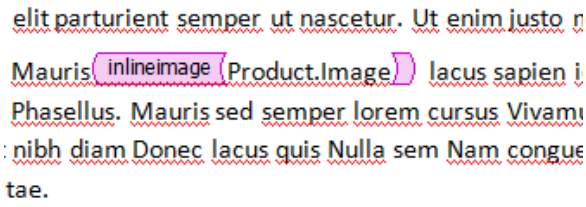
OBRÁZOK 23:ZNAČKA SWITCH

Include

Pomocou tejto značky sa dá na špecifikované miesto vložiť iný Word dokument. Cesta k vkladanému dokument je špecifikovaná buď atribútom *src* alebo *file*. Do atribútu *src* sa ukladá názov vlastnosti dátového zdroja, ktorý obsahuje cestu k súboru, do atribútu *file* sa vkladá statická cesta k súboru. V prípade že sú vyplnené oba, berie sa prioritne hodnota atribútu *src*.

Inlineimage

Obrázky vkladané do Word dokumentov, sa delia na dve skupiny – tzv. *inline* obrázky, ktoré sú súčasťou toku dokumentu, alebo *ukotvené* obrázky ktoré majú svoju pozíciu špecifikovnú relatívne k nejakému elementu. Značka *inlineimage* predstavuje prvý typ. Obrázok môže byť špecifikovaný cestou v súborovom systéme (atribút *path* – odkazuje sa na vlastnosť dátového zdroja, kde je cesta uložená) alebo môže dátový zdroj obsahovať priamo vlastnosť ktorá vráti obrázok ako pole bajtov. V tomto prípade je názov vlastnosti daný textom, ktorý je obalený elementom. Atribútmi *width* a *height* sa určuje veľkosť obrázka v centimentroch.



OBRÁZOK 24:ZNAČKA INLINEIMAGE

Anchorimage

Táto značka predstavuje obrázok ukotvený k nejakému elementu v dokumente (odsek, strana, okraje atď.) Cesta k obrázku je definovaná rovnakým spôsobom ako u značky *inlineimage* (teda buď atribútom *path* alebo obsahom elementu). Táto značka má však širšie možnosti nastavenia vlastností obrázku. Vertikálna pozícia sa špecifikuje buď atribútom *left* alebo *v-align*, horizontálna pozícia sa určuje pomocou atribútu *top* alebo *h-align*

Atribút	Hodnoty	Popis
width	číslo	šírka v cm
height	číslo	výška v cm
left	číslo	vzdialenosť od ľavého okraja kotviaceho elementu v bodoch
top	číslo	vzdialenosť od horného okraja kotviaceho elementu v bodoch
v-align	• Center • Inside • Top • Outside • Bottom	Relatívna vertikálna pozícia v rámci kotviaceho elementu.
h-align	• Center • Inside • Left • Outside	Relatívna horizontálna pozícia v rámci kotviaceho elementu.

	• Right	
text-wrap	• None • Square • Topbottom	Zalomenie textu okolo obrázka.
v-pos-relative-from	• Topmargin • Bottommargin • Page • Insidemargin • Outsidemargin • Margin • Line • Paragraph	Určuje elememt dokumentu, voči ktorému sa vzťahuje vertikálna pozícia.
h-pos-relative-from	• Leftmargin • Rightmargin • Page • Column • Character • Insidemargin • Outsidemargin • Margin	Určuje elememt dokumentu, voči ktorému sa vzťahuje vertikálna pozícia.

TABUĽKA 6:ATRIBÚTY UKOTVENÉHO OBRÁZKA

Placeholder

Značka *placeholder* neslúži na generovanie dokumentov, ale určuje miesto v dokumente na ktoré sa dá odkazovať pri vkladaní textových polí alebo obrázkov (viď. Kapitola 4.8). Tento element musí mať určené svoje meno, jednoznačné v rámci dokumentu, pomocou atribútu *name*.

Customtag

Značka predstavuje rozšírenie funkcionality, viď. kapitola 4.7.

C.3 Príklad použitia komponenty

Vo výpise 9 je uvedený jednoduchý príklad použitia komponenty.

Pri generovaní dokumentu je potrebné vytvoriť inštanciu triedy *Document*, ktorá potrebuje poznať objekt šablóny (*Template*) a dátovy zdroj (*DataSource*). Šablóna musí mať nastavený objekt ktorý implementuje *IParser* (v našom prípade je to inštancia triedy *WordParser*) a cestu k súboru zo šablónou.

```
//Získame dátový zdroj v podobe objektu - dataObject(úloha dátovej vrstvy
aplikácie)
Document doc = new Document();

//vytvorenie šablóny - konštruktore jej predáme inštanciu IParser-a
Template test = new Template(new WordParser()) { TemplatePath =
                    @"d:\projekty\TemplateParser\sablona3.docx" };
doc.Template = test;

//priradenie dátového zdroja dokumentu
doc.DataSource = dataObject;

doc.ParseDocumentFromTemplate(@"d:\projekty\TemplateParser\parsed-sablona.docx");
```

VÝPIS 9:PRÍKLAD POUŽITIA KOMPONENTY